Multiplex Interface (MXI) Specification

Multiplex Interface (MXI) Specification

Version 1.1 Edition 7.20141001
Updated October 25, 2014
Distributed with Package openss7-1.1.7.20141001

Abstract:

This document is a Specification containing technical details concerning the implementation of the Multiplex Interface (MXI) for OpenSS7. It contains recommendations on software architecture as well as platform and system applicability of the Multiplex Interface (MXI). It provides abstraction of the Multiplex (MX) interface to these components as well as providing a basis for Multiplex control for other Multiplex 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:

OpenSS7 Corporation disclaims all warranties with regard to this documentation including all implied warranties of merchantability, fitness for a particular purpose, non-infringement, or title; that the contents of the document are suitable for any purpose, or that the implementation of such contents will not infringe on any third party patents, copyrights, trademarks or other rights. In no event shall OpenSS7 Corporation be liable for any direct, indirect, special or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with any use of this document or the performance or implementation of the contents thereof.

Short Contents

Table of Contents

List of Figures

List of Tables

Preface

Notice

Abstract

This document is a Specification containing technical details concerning the implementation of the Multiplex Interface (MXI) for OpenSS7. It contains recommendations on software architecture as well as platform and system applicability of the Multiplex Interface (MXI).

This document specifies a Multiplex Interface (MXI) Specification in support of the OpenSS7 Multiplex (MX) protocol stacks. It provides abstraction of the Multiplex interface to these components as well as providing a basis for Multiplex control for other Multiplex protocols.

Purpose

The purpose of this document is to provide technical documentation of the Multiplex Interface (MXI). 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 Multiplex Interface (MXI) 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 Multiplex Interface (MXI). This document is intended to provide information for writers of OpenSS7 Multiplex Interface (MXI) applications as well as writers of OpenSS7 Multiplex Interface (MXI) Users.

Audience

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

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.

A current version of this specification is normally distributed with the *OpenSS7* package, `openss7-1.1.7.20141001`.[1]

[1] `http://www.openss7.org/repos/tarballs/openss7-1.1.7.20141001.tar.bz2`

Version Control

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. *OpenSS7 Corporation* is making this documentation available as a reference point for the industry. While *OpenSS7 Corporation* believes that these interfaces are well defined in this release of the document, minor changes may be made prior to products conforming to the interfaces being made available. *OpenSS7 Corporation* reserves the right to revise this software and documentation for any reason, including but not limited to, conformity with standards promulgated by various agencies, utilization of advances in the state of the technical arts, or the reflection of changes in the design of any techniques, or procedures embodied, described, or referred to herein. *OpenSS7 Corporation* is under no obligation to provide any feature listed herein.

```
$Log: mxi.texi,v $
Revision 1.1.2.2  2011-02-07 02:21:41  brian
- updated manuals

Revision 1.1.2.1  2009-06-21 10:54:32  brian
- added files to new distro
```

ISO 9000 Compliance

Only the TEX, texinfo, or roff source for this maual is controlled. An opaque (printed, postscript or portable document format) version of this manual is a **UNCONTROLLED VERSION**.

Disclaimer

OpenSS7 Corporation **disclaims all warranties with regard to this documentation including all implied warranties of merchantability, fitness for a particular purpose, non-infrincement, or title; that the contents of the manual are suitable for any purpose, or that the implementation of such contents will not infringe on any third party patents, copyrights, trademarks or other rights. In no event shall** *OpenSS7 Corporation* **be liable for any direct, indirect, special or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action or contract, negligence or other tortious action, arising out of or in connection with any use of this documentation or the performance or implementation of the contents thereof.**

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 Aquisition Regulations ("DFARS") (or any successor regulations) and the Government is acquiring only the license rights granded 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 Supplerment to the FAR (or any successor regulations).

Acknowledgements

The OpenSS7 Project was funded in part by:

- Monavacon Limited
- OpenSS7 Corporation

Thanks to the subscribers to and sponsors of The OpenSS7 Project. Without their support, open software like this would not be possible.

As with most open source projects, this project would not have been possible without the valiant efforts and productive software of the Free Software Foundation, the Linux Kernel Community, and the open source software movement at large.

1 Introduction

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

This specification assumes that the reader is familiar with ITU-T state machines and multiplex interface (e.g. G.703, G.704), and STREAMS.

1.1 Related Documentation

- **ITU-T Recommendation G.703 (White Book)**
- **ITU-T Recommendation G.704 (White Book)**
- **ANSI T1**
- **System V Interface Definition, Issue 2 - Volume 3**

1.1.1 Role

This document specifies an interface that supports the services provided by the *Multiplex* for ITU-T, ANSI and ETSI applications as described in ITU-T Recommendation G.703 and ITU-T Recommendation G.704. These specifications are targeted for use by developers and testers of protocol modules that require multiplex 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.

ISO International Organization for Standardization

OSI Open Systems Interconnection

QOS Quality of Service

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

2 The Multiplex Layer

The Multiplex Layer provides the means to manage the association of MX-Users info connections. It is responsible for the routing and management of data to and from multiplex connections between MX-user entities.

2.1 Model of the MXI

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

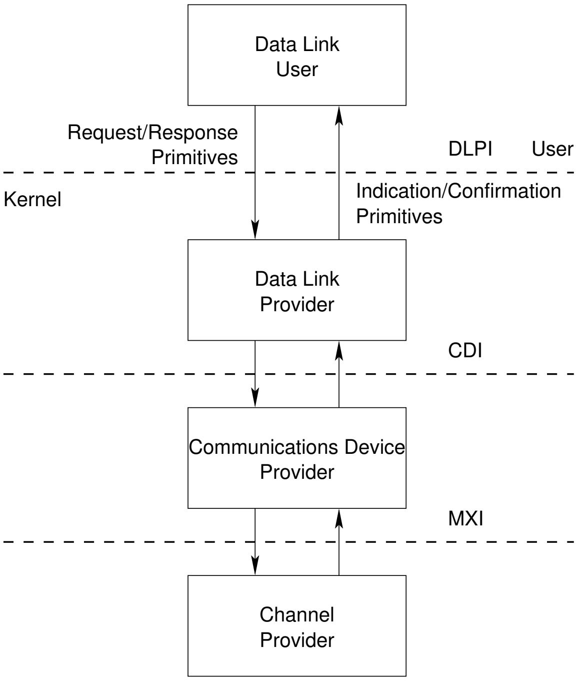
Data Link User
Request/Response Primitives
DLPI
User
Kernel
Indication/Confirmation Primitives
Data Link Provider
CDI
Communications Device Provider
MXI
Channel Provider

Figure 2.1: *Model of the MXI*

The MXI allows the MXS provider to be configured with any multiplex layer user (such as a signalling data terminal application) that also conforms to the MXI. A multiplex layer user can also be a user program that conforms to the MXI and accesses the MXS provider via **putmsg**(2s) and **getmsg**(2s) system calls. The typical configuration, however, is to place a signalling data terminal module above the multiplex layer.

2.2 MXI Services

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

The MXI 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 connecting a Stream to a medium, exchanging bits with the medium, and disconnecting the Stream from the medium.

2.2.1 Local Management

Local management services are listed in Table 2.1.

Phase	Service	Primitives
Local Management	Acknowledgement	MX_OK_ACK, MX_ERROR_ACK
	Information Reporting	MX_INFO_REQ, MX_INFO_ACK
	PPA Attachment	MX_ATTACH_REQ, MX_DETACH_REQ, MX_OK_ACK
	Initialization	MX_ENABLE_REQ, MX_ENABLE_CON, MX_DISABLE_REQ, MX_DISABLE_CON
	Options Management	MX_OPTMGMT_REQ, MX_OPTMGMT_ACK
	Event Reporting	MX_ERROR_IND, MX_STATS_IND, MX_EVENT_IND

Table 2.1: *Local Management Services*

The local management services interface is described in Section 3.1 [Local Management Services], page 15, and the primitives are detailed in Section 4.1 [Local Management Service Primitives], page 25. The local management services interface is defined by the `sys/mxi.h` header file (see Appendix A [MXI Header Files], page 75).

2.2.2 Protocol

Protocol services are listed in Table 2.2.

Phase	Service	Primitives
Protocol	Connection	MX_CONNECT_REQ
	Data Transfer	MX_DATA_REQ, MX_DATA_IND
	Disconnection	MX_DISCONNECT_REQ, MX_DISCONNECT_IND

Table 2.2: *Protocol Services*

The protocol services interface is described in Section 3.2 [Protocol Services], page 21, and the primitives are detailed in Section 4.2 [Protocol Service Primitives], page 50. The protocol services interface is defined by the `sys/mxi.h` header file (see Appendix A [MXI Header Files], page 75).

2.3 Purpose of the MXI

The MXI is typically implemented as a device driver controlling a TDM (Time Division Mutliplexing) device that provides access to multiplexes. The purpose behind exposing this low level interface is that almost all communications multiplex devices can be placed into a *raw* mode, where a bit stream can be exchanged between the driver and the medium. The MXI provides an interface that, once implemented as a driver for a new device, can provide complete and verified data link capabilities by pushing generic HDLC (High Level Data Link Control) and LAPB (Link Access Procedure Balanced) modules over an open device Stream.

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

2.4 Multiplex Addressing

Each use of MXI must establish an identity to communicate with other multiplex users. The MXS user must identify the physical medium over which it wil communicate. This is particularly evident on system that are attached to multiple physical media. Figure 2.2 illustrates the identification approach, which is explained below.

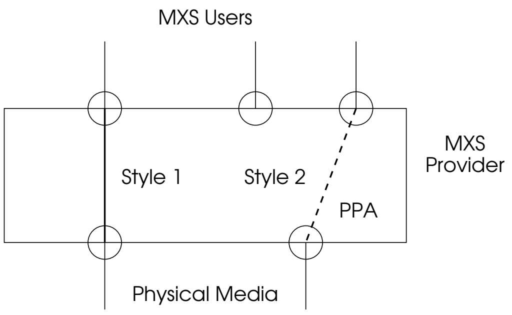

Figure 2.2: *Multiplex Addressing Components*

2.4.1 Physical Attachment Identification

The physical point of attachment (PPA in Figure 2.2) is the point at which a system interface attaches itself to a physical communications medium (a channel, facility or network interface). All communication on that physical medium funnels through the PPA associated with that physical medium. On systems where a MXS provider supports more than on physical medium, the MXS user must identify the medium through which it will communicate. A PPA is identified by a unique PPA identifier.

For media that supports physical layer multiplexing of multiple channels over a single physical medium (such as the B and D channels of ISDN), the PPA identifier must identify the specific channel(s) over which communication will occur. See also [Multiplex Media], page 12.

Unlike the Data Link Provider Interface (DLPI), which also uses the concept of a PPA, MXI does not define a SAP for a MXS user.

Once a Stream has been associated with a PPA, all messages received on that medium are delivered to the attached MXS user. Only one major/minor device number combination (Stream head) can be associated with a given PPA and active for a range of channels at any point in time.

2.4.2 MXS Provider Styles

Two styles of MXS provider are defined by MXI, distinguished by the way they enable a MXS user to choose a particular PPA.

2.4.2.1 Style 1 MXS Provider

The *Style 1* provider assigns a PPA based on the major/minor device the MXS user opened. One possible implementation of a *Style 1* driver would reserve a major device for each PPA the multiplex device driver would support. This would allos the STREAMS clone open feature to be used for each PPA configured. This style of provider is appropriate when few PPAs will be supported.

For example, a CPI card that supports two V.35 ports could assign a major device number to the card diver and a minor device number to each of the ports on each card in the system. To establish a Stream to a MXS provider for a given port, the minor device number '`1`' or '`2`' could be opened for port '`1`' or '`2`' on card '`1`', minor device number '`3`' or '`4`' could be opened for port '`1`' or '`2`' on card '`2`', and so on. One major device number for the driver could easily support 127 cards in a system, which is not possible for typical PCI systems and, therefore, is ample.

Style 1 providers do not user the `MX_ATTACH_REQ` and `MX_DETACH_REQ` primitives and when freshly opened are in the `MXS_ATTACHED` state. That is, as illustrated in Figure 2.2, the *Style 1* MXS provider associates the Stream with the PPA during the `open(2s)` system call.

2.4.2.2 Style 2 MXS Provider

If the number of PPAs as MXS provider will support is large, a *Style 2* provider implementation is more suitable. The *Style 2* provider requires a MXS user to explicitly identify the desired PPA using a special attach service primitive. For a *Style 2* driver, the `open(2s)` system call creates a Stream between the MXS user and MXS provider, and the attach primitive then associated a particular PPA with that Stream. The format of the PPA identifier is specific to the MXS provider, and should be described in the provider-specific addendum documentation.

The MXS user uses the support primitvies(`MX_ATTACH_REQ`, `MX_ENABLE_REQ`) to associate a Stream with a given Physical Point of Appearance. *Style 2* MXS providers, when freshly opened, are in the `MXS_DETACHED` state. That is, the *Style 2* MXS provider does not associate the Stream with the PPA during the `open(2s)` call, but only later when the `MX_ATTACH_REQ` primitive is issued by the MXS user.

2.4.3 Multiplex Media

To accommodate multiplexed media and multi-media channels, there are three kinds of PPA address:

1. A discrete PPA that specifies a non-multiplexed medium.

   This is the normal case of a *Style 1* or *Style 2* MXS provider supporting access to a non-multiplexed medium. An example is a MXS provider supporting access to a V.35 interface.

2. A specific PPA that specifies a single channel to a multiplexed medium.

   This is again the normal case of a *Style 1* or *Style 2* MXS provider supporting access to a specific channel in a multiplexed medium. An example is a MXS provider supporting access to channel 16 of a E1 interface.

3. A general PPA that specifies a channel group for a multiplexed medium.

   This is th case of a *Style 1* or *Style 2* MXS provider supporting access to multiple channels in a multiplexed medium. An example is a MXS provider supporting statistically multiplexed channel access to a full or fractional T1 facilitiy. Another example is access to the left and right channels of a stereo program.

In the case of a general PPA, as enumerated in *3* above, some additional information is required to identify which slots in the group of channle forming the multiplex are associatedw the the MXS user Stream. This additional information is provided using the *mx_slot* parameter to the `MX_CONNECT_REQ`, `MX_CONNECT_CON`, `MX_DATA_REQ`, `MX_DATA_IND`, `MX_EVENT_IND`, `MX_DISCONNECT_REQ`, `MX_DISCONNECT_CON` and `MX_DISCONNECT_IND` primitives.[1]

2.5 Multiplex Parameters

[1] Note that it is the ability of the Multiplex Interface to support fractional E1/T1 that distinguishes it from similar interfaces such as the SDLI and CDI.

3 MXI Services Definition

3.1 Local Management Services

3.1.1 Acknowledgement Service

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

- `MX_OK_ACK`: The `MX_OK_ACK` message is used by the MXS provider to indicate successful receipt and completion of a service primitive request that requires positive acknowledgement.
- `MX_ERROR_ACK`: The `MX_ERROR_ACK` message is used by the MXS 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.

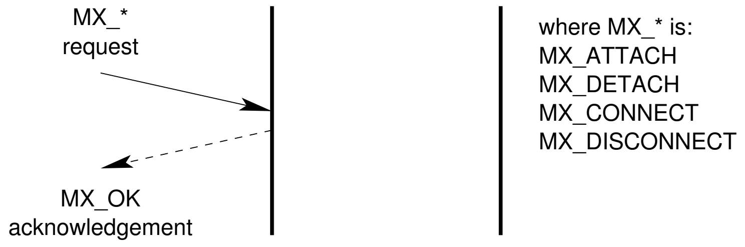

Figure 3.1: *Message Flow: Successful Acknowledgement Service*

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

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

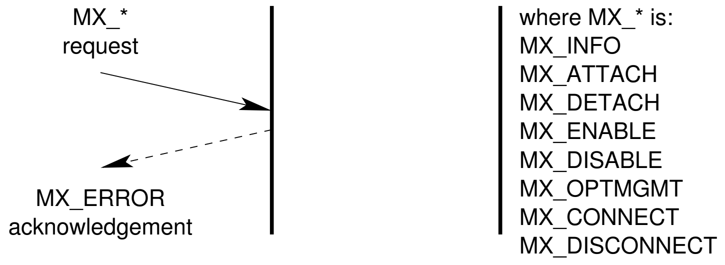

Figure 3.2: *Message Flow: Unsuccessful Acknowledgement Service*

As illustrated in Figure 3.2, the service primitives for which a negative acknowledgement may be returned are the `MX_INFO_REQ`, `MX_ATTACH_REQ`, `MX_DETACH_REQ`, `MX_ENABLE_REQ`, `MX_DISABLE_REQ` and `MX_OPTMGMT_REQ` messages.

3.1.2 Information Reporting Service

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

- `MX_INFO_REQ`: The `MX_INFO_REQ` message is used by the MXS user to request information about the MXS provider.
- `MX_INFO_ACK`: The `MX_INFO_ACK` message is issued by the MXS provider to provide requested information about the MXS provider.

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

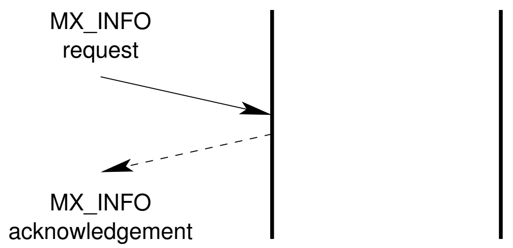

Figure 3.3: *Message Flow: Successful Information Reporting Service*

3.1.3 Physical Point of Attachment Service

The local management interface provides the MXS 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 MXS provider:[1]

Style 1 MXS Provider

A *Style 1* MXS provider is a provider that associates a Stream with a PPA at the time of the first `open(2s)` call for the device, and disassociates a Stream from a PPA at the time of the last `close(2s)` 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* MXS provider Streams start life in the `MX_DISABLED` state.

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

[1] See also Section 2.4 [Multiplex Addressing], page 11.

Style 2 MXS Provider

A *Style 2* MXS provider is a provider that associates a Stream with a PPA at the time that the MXS user issues the `MX_ATTACH_REQ` message. Freshly opened Streams are not associated with any PPA. The *Style 2* MXS provider Stream is disassociated from a PPA when the Stream is closed or when the MXS user issues the `MX_DETACH_REQ` message.

Freshly opened *Style 2* MXS provider Streams start life in the `MX_UNATTACHED` state.

This approach is suitable for MXS 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 MXS user with the ability to attach a *Style 2* MXS provider Stream to a physical point of appearance (PPA).

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

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

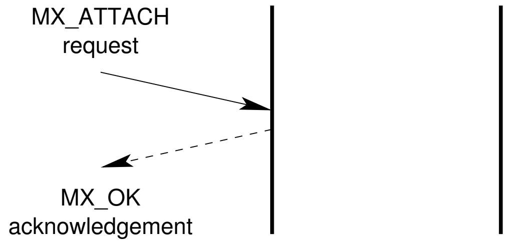

Figure 3.4: *Message Flow: Successful Attachment Service*

3.1.3.2 PPA Detachment Service

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

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

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

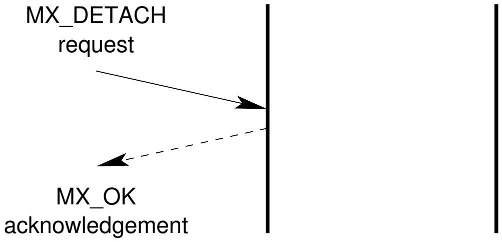

Figure 3.5: *Message Flow: Successful Detachment Service*

3.1.4 Initialization Service

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

- `MX_ENABLE_REQ`: The `MX_ENABLE_REQ` message is issued by the MXS user to request that the protocol service interface be enabled.
- `MX_ENABLE_CON`: Upon successful enabling of the protocol service interface, the MXS provider acknowledges successful completion of the service by issuing a `MX_ENABLE_CON` message to the MXS user.
- `MX_ERRORK_ACK`: Upon unsuccessful enabling of the protocol service interface, the MXS provider acknowledges the failure to complete the service by issuing an `MX_ERROR_ACK` message to the MXS user.

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

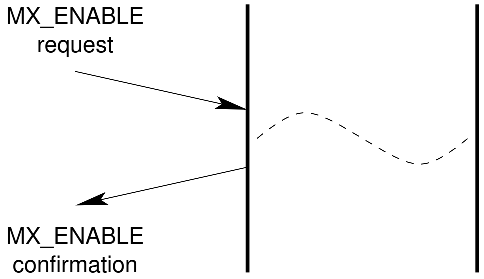

Figure 3.6: *Message Flow: Successful Enable Service*

3.1.4.2 Interface Disable Service

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

- `MX_DISABLE_REQ`: The `MX_DISABLE_REQ` message is issued by the MXS user to request that the protocol service interface be disabled.
- `MX_DISABLE_CON`: Upon successful disabling of the protocol service interface, the MXS provider acknowledges successful completion of the service by issuing a `MX_DISABLE_CON` message to the MXS user.
- `MX_ERRORK_ACK`: Upon unsuccessful disabling of the protocol service interface, the MXS provider acknowledges the failure to complete the service by issuing an `MX_ERROR_ACK` message to the MXS user.
- `MX_DISABLE_IND`: The `MX_DISABLE_IND` message is used by the MXS provider to indicate to the MXS user that the Stream has been autonomously disabled and the cause of the autonomous disabling.

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

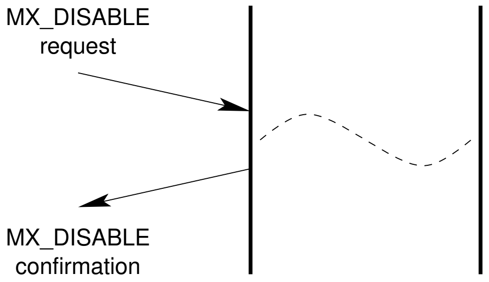

Figure 3.7: *Message Flow: Successful Disable Service*

3.1.5 Options Management Service

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

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

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

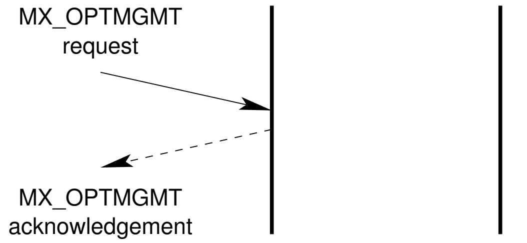

Figure 3.8: *Message Flow: Successful Options Management Service*

3.1.6 Error Reporting Service

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

- `MX_ERROR_IND`: The MXS provider issues the `MX_ERROR_IND` message to the MXS 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.

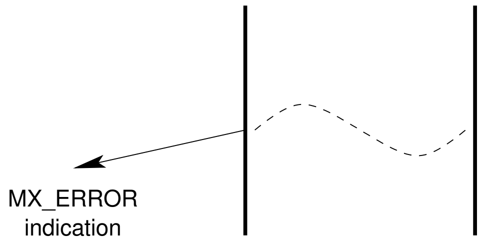

Figure 3.9: *Message Flow: Successful Error Reporting Service*

3.1.7 Statistics Reporting Service

- `MX_STATS_IND`:

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

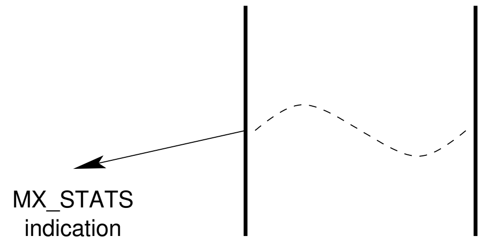

Figure 3.10: *Message Flow: Successful Statistics Reporting Service*

3.1.8 Event Reporting Service

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

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

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

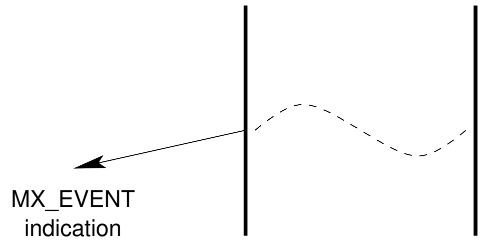

Figure 3.11: *Message Flow: Successful Event Reporting Service*

3.2 Protocol Services

Protocol services are specific to the Multiplex interface. These services consist of connection services that permit the transmit and receive directions to be connected to or disconnected from the medium, and data transfer services that permit the exchange of bits between MXS users.

The service primitives that implement the protocol services are described in detail in Section 4.2 [Protocol Service Primitives], page 50.

3.2.1 Connection Service

The connection service provides the ability for the MXS user to connect to the medium for the purpose of transmitting bits, receiving bits, or both. In the OSI model, this is a Layer 1 function, possibly the responsibility of multiplex or digital cross-connect switch.

- MX_CONNECT_REQ: The MX_CONNECT_REQ message is used by the MXS user to request that the Stream be connected to the medium. Connection to the medium might require some switching or other mechanism to prepare the Stream for data transmission and reception. Connections can be formed for the receive direction or the transmit direction independently.
- MX_CONNECT_CON: The MX_CONNECT_CON message is used by the MXS provider to confirm that the Stream has been connected to the medium. Connection to the medium may have required some switching or other mechanism to prepare the Stream for data transmission and receptoin. Connection can be confirmed for the receive or transmit directions independently.

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

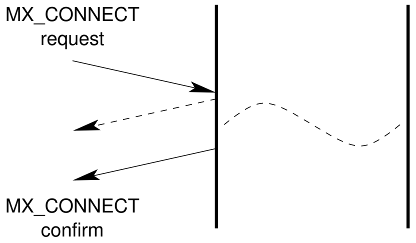

Figure 3.12: *Message Flow: Successful Connection Service*

3.2.2 Data Transfer Service

The data transfer service provides the MXS user with the ability to request that bits be transmitted on the medium, and the MXS provider with the ability to indicate bits that have been received from the medium.

- MX_DATA_REQ: The MX_DATA_REQ message is used by the MXS user to place raw bits onto the medium. The Stream must have first been successfully activated in the transmit direction using the MX_CONNECT_REQ message.
- MX_DATA_IND: The MX_DATA_IND message is issued by the MXS provider when activated for the receive direction with the MX_CONNECT_REQ message, to indicate bits received on the medium.

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

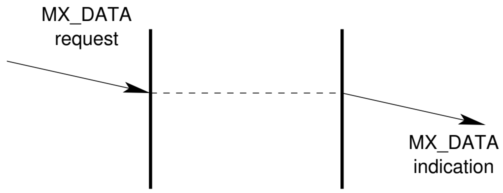

Figure 3.13: *Message Flow: Successful Data Transfer Service*

3.2.3 Disconnection Service

The disconnection service provides the ability for the MXS user to disconnect from the medium, withdrawing from the purpose of transmitting bits, receiving bits, or both. It allows the MXS provider to autonomously indicate that the medium has been disconnected from the Stream. In OSI, this is a Layer 1 function, possibly the responsibilyt of a multiplex or digital cross-connect switch.

- `MX_DISCONNECT_REQ`: The `MX_DISCONNECT_REQ` message is used by the MXS user to request that the Stream be disconnected from the medium. Disconnection from the medium might require some switching or other mechanism. Disconnection can be performed for the receive direction or the transmit direction independently.
- `MX_DISCONNECT_CON`: The `MX_DISCONNECT_CON` message is used by the MXS provider to confirm that the Stream has been disconnected from the medium. Disconnect from the medium might require some switching or other mechanism. Disconnection can be confirmed for the receive or transmit directions independently.
- `MX_DISCONNECT_IND`: The `MX_DISCONNECT_IND` message is used by the MXS provider to indicate to the MXS user that the Stream has been disconnected from the medium. Disconnection is indicated for both the receive and transmit directions.

A successful invocation of the disconnection service by the MXS user is illustrated in Figure 3.14.

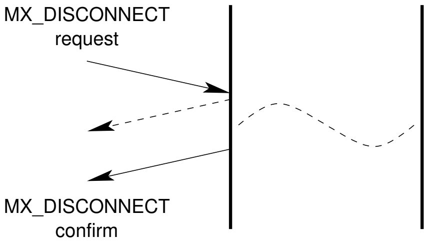

Figure 3.14: *Message Flow: Successful Disconnection Service by SDLS User*

A successful invocation of the disconnection service by the MXS provider is illustrated in Figure 3.15.

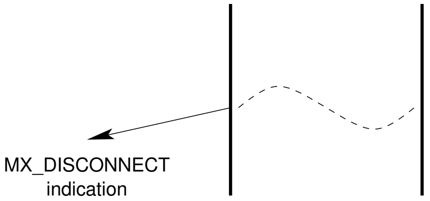

Figure 3.15: *Message Flow: Successful Disconnection Service by SDLS Provider*

4 MXI Service Primitives

4.1 Local Management Service Primitives

These service primitives implement the local management services (see Section 3.1 [Local Management Services], page 15).

4.1.1 Acknowledgement Service Primitives

These service primitives implement the acknowledgement service (see Section 3.1.1 [Acknowledgement Service], page 15).

4.1.1.1 MX_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 MX_ok_ack {
    mx_ulong mx_primitive;
    mx_ulong mx_correct_prim;
    mx_ulong mx_state;
} MX_ok_ack_t;
```

Parameters

The service primitive contains the following parameters:

mx_primitive

Indicates the service primitive type. Always `MX_OK_ACK`.

mx_correct_prim

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

`MX_ATTACH_REQ`	Attach request.
`MX_ENABLE_REQ`	Enable request.
`MX_CONNECT_REQ`	Connect request.
`MX_DISCONNECT_REQ`	Disconnect request.
`MX_DISABLE_REQ`	Disable request.
`MX_DETACH_REQ`	Detach Request.

mx_state

Indicates the current state of the MXS provider at the time that the primitive was issued. This field can be one of the following values;

`MXS_UNINIT`	Unitialized.
`MXS_UNUSABLE`	Device cannot be used, Stream in hung state.
`MXS_DETACHED`	No PPA attached, awaiting `MX_ATTACH_REQ`.

MXS_ATTACHED	PPA attached, awaiting MX_ENABLE_REQ.
MXS_WCON_EREQ	Waiting to send MX_ENABLE_CON.
MXS_WCON_RREQ	Waiting to send MX_DISABLE_CON.
MXS_ENABLED	Ready for use, awaiting primitive exchange.
MXS_WCON_CREQ	Waiting to send MX_CONNECT_CON.
MXS_WCON_DREQ	Waiting to send MX_DISCONNECT_CON.
MXS_CONNECTED	Connected, active data transfer.

State

This primitive is issued by the MXS provider in the MXS_WACK_AREQ, MXS_WACK_UREQ, MXS_WACK_CREQ or MXS_WACK_DREQ state.

New State

The new state is MXS_DETACHED, MXS_ATTACHED, MXS_ENABLED or MXS_CONNECTED, depending on the primitive to which the message is responding.

4.1.1.2 MX_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 MX_error_ack {
    mx_ulong mx_primtive;
    mx_ulong mx_error_primitive;
    mx_ulong mx_error_type;
    mx_ulong mx_unix_error;
    mx_ulong mx_state;
} MX_error_ack_t;
```

Parameters

The error acknowledgement primitive contains the following parameters:

mx_primitive

Indicates the primitive type. Always `MX_ERROR_ACK`.

mx_error_type

Indicates the MX error number. This field can have one of the following values:

[MXSYSERR]	UNIX system error.
[MXBADADDR]	Bad address format or content.
[MXOUTSTATE]	Interface out of state.
[MXBADOPT]	Bad options format or content.
[MXBADPARM]	Bad parameter format or content.
[MXBADPARMTYPE]	Bad paramater structure type.
[MXBADFLAG]	Bad flag.
[MXBADPRIM]	Bad primitive.
[MXNOTSUPP]	Primitive not supported.
[MXBADSLOT]	Bad multplex slot.

mx_unix_error

Indicates the reason for failure. This field is protocol-specific. When the *mx_error_type* field is `[MXSYSERR]`, the *mx_unix_error* field is the UNIX error number as described in `errno(3)`.

mx_error_primitive

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

MX_INFO_REQ	Information request.
MX_OPTMGMT_REQ	Options management request.
MX_ATTACH_REQ	Attach request.
MX_ENABLE_REQ	Enable request.
MX_CONNECT_REQ	Connect request.
MX_DATA_REQ	Data request.
MX_DISCONNECT_REQ	Disconnect request.
MX_DISABLE_REQ	Disable request.

`MX_DETACH_REQ`	Detach Request.
`MX_INFO_ACK`	Information acknowledgement.
`MX_OPTMGMT_ACK`	Options Management acknowledgement.
`MX_OK_ACK`	Successful receipt acknolwedgement.
`MX_ERROR_ACK`	Error acknowledgement.
`MX_ENABLE_CON`	Enable confirmation.
`MX_CONNECT_CON`	Connect confirmation.
`MX_DATA_IND`	Data indication.
`MX_DISCONNECT_IND`	Disconnect indication.
`MX_DISCONNECT_CON`	Disconnect confirmation.
`MX_DISABLE_IND`	Disable indication.
`MX_DISABLE_CON`	Disable confirmation.
`MX_EVENT_IND`	Event indication.

mx_state

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

`MXS_UNINIT`	Unitialized.
`MXS_UNUSABLE`	Device cannot be used, Stream in hung state.
`MXS_DETACHED`	No PPA attached, awaiting `MX_ATTACH_REQ`.
`MXS_WACK_AREQ`	Waiting for attach.
`MXS_WACK_UREQ`	Waiting for detach.
`MXS_ATTACHED`	PPA attached, awaiting `MX_ENABLE_REQ`.
`MXS_WCON_EREQ`	Waiting to send `MX_ENABLE_CON`.
`MXS_WCON_RREQ`	Waiting to send `MX_DISABLE_CON`.
`MXS_ENABLED`	Ready for use, awaiting primitive exchange.
`MXS_WACK_CREQ`	Waiting acknolwedgement of `MX_CONNECT_REQ`.
`MXS_WCON_CREQ`	Waiting to send `MX_CONNECT_CON`.
`MXS_WACK_DREQ`	Waiting acknolwedgement of `MX_DISCONNECT_REQ`.
`MXS_WCON_DREQ`	Waiting to send `MX_DISCONNECT_CON`.
`MXS_CONNECTED`	Connected, active data transfer.

State

This primitive can be issued in any state for which a local acknowledgement is not pending. The MXS 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 16).

4.1.2.1 MX_INFO_REQ

Description

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

Format

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

```
typedef struct MX_info_req {
    mx_ulong mx_primitive;
} MX_info_req_t;
```

Parameters

This primitive contains the following parameters:

mx_primitive
Specifies the primitive type. Always MX_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 MXS provider to acknowledge receipt of the primitive as follows:

- **Successful**: The MXS provider is required to acknowledge receipt of the primitive and provide the requested information using the MX_INFO_ACK primitive.
- **Unsuccessful (non-fatal errors)**: The MXS provider is required to negatively acknowledge the primtive using the MX_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:

[MXSYSERR]	UNIX system error.
[MXBADADDR]	Bad address format or content.
[MXOUTSTATE]	Interface out of state.
[MXBADOPT]	Bad options format or content.
[MXBADPARM]	Bad parameter format or content.
[MXBADPARMTYPE]	Bad paramater structure type.
[MXBADFLAG]	Bad flag.
[MXBADPRIM]	Bad primitive.
[MXNOTSUPP]	Primitive not supported.
[MXBADSLOT]	Bad multplex slot.

4.1.2.2 MX_INFO_ACK

Description

This MXS provider originated primitive acknowledges receipt and successful processing of the MX_INFO_REQ primitive and provides the requested information concerning the MXS provider.

Format

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

```
typedef struct MX_info_ack {
    mx_ulong mx_primitive;    /* always MX_INFO_ACK */
    mx_ulong mx_addr_length;  /* multiplex address length */
    mx_ulong mx_addr_offset;  /* multiplex address offset */
    mx_ulong mx_parm_length;  /* multiplex paramters length */
    mx_ulong mx_parm_offset;  /* multiplex paramters offset */
    mx_ulong mx_prov_flags;   /* provider options flags */
    mx_ulong mx_prov_class;   /* provider class */
    mx_ulong mx_style;        /* provider style */
    mx_ulong mx_version;      /* multiplex interface version */
    mx_ulong mx_state;        /* multiplex state */
} MX_info_ack_t;
```

Parameters

The information acknowledgement service primitive has the following parameters:

mx_primitive
: Indicates the service primitive type. Always MX_INFO_ACK.

mx_addr_length
: Indicates the length of the PPA address to which the provider is attached. When in states MXS_DETACHED or MXS_WACK_AREQ, this value will be zero ('0').

mx_addr_offset
: Indicates the offset, beginning from the start of the M_PCPROTO message block of the PPA address associated with the provider. When the *mx_addr_length* field is zero, this field is also zero ('0').

mx_parm_length
: Indicates the length of the parameters associated with the provider.

mx_parm_offset
: Indicates the offset, beginning from the start of the M_PCPROTO message block, of the parameters associated with the provider. When the *mx_parm_length* field is zero, this field is also zero ('0').

mx_prov_flags
: Indicates the options flags associated with the provider. This is a bitwise OR of zero or more of the following flags:

mx_prov_class
: Indicates the provider class. This can be one of the following values:

MX_CIRCUIT	Circuit provider class.

mx_addr_length

This is a variable length field. The length of the field is determined by the length attribute.

For a *Style 2* driver, when *mx_style* is `MX_STYLE2`, and when in an attached state, this field provides the current PPA associated with the Stream; the length is typically 4 bytes.

For a *Style 1* driver, when *mx_ppa_stype* is `MX_STYLE1`, the length is 0 bytes.

mx_style Indicates the PPA style of the MXS provider. This value can be one of the following values;

`MX_STYLE1`	PPA is implicitly attached by `open(2s)`.
`MX_STYLE2`	PPA must be explicitly attached using `MX_ATTACH_REQ`.

mx_version The version of the interface. This version is `MX_VERSION_1_1`.

`MX_VERSION_1_0`	Version 1.0 of interface.
`MX_VERSION_1_1`	Version 1.1 of interface.
`MX_VERSION`	Always the current version of the header file.

mx_state Indicates the state of the MXS provider at the time that the information acknolwedgement service primitive wsa issued. This field can be one of the following values:

`MXS_UNINIT`	Unitialized.
`MXS_UNUSABLE`	Device cannot be used, Stream in hung state.
`MXS_DETACHED`	No PPA attached, awaiting `MX_ATTACH_REQ`.
`MXS_WACK_AREQ`	Waiting for attach.
`MXS_WACK_UREQ`	Waiting for detach.
`MXS_ATTACHED`	PPA attached, awaiting `MX_ENABLE_REQ`.
`MXS_WCON_EREQ`	Waiting to send `MX_ENABLE_CON`.
`MXS_WCON_RREQ`	Waiting to send `MX_DISABLE_CON`.
`MXS_ENABLED`	Ready for use, awaiting primitive exchange.
`MXS_WACK_CREQ`	Waiting acknolwedgement of `MX_CONNECT_REQ`.
`MXS_WCON_CREQ`	Waiting to send `MX_CONNECT_CON`.
`MXS_WACK_DREQ`	Waiting acknolwedgement of `MX_DISCONNECT_REQ`.
`MXS_WCON_DREQ`	Waiting to send `MX_DISCONNECT_CON`.
`MXS_CONNECTED`	Connected, active data transfer.

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 16).

4.1.3.1 MX_ATTACH_REQ

Description

This MXS user originated primitive requests that the Stream upon which the primitive is issued be associated with the specified Physical Point of Attachment (PPA). This primitive is only applicable to *Style 2* MXS provider Streams, that is, Streams that return `MX_STYLE2` in the *mx_style* field of the `MX_INFO_ACK`.

Format

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

```
typedef MX_attach_req {
    mx_ulong mx_primitive;
    mx_ulong mx_addr_length;
    mx_ulong mx_addr_offset;
    mx_ulong mx_flags;
} MX_attach_req_t;
```

Parameters

The attach request primitive contains the following parameters:

mx_primitive
Specifies the service primitive type. Always `MX_ATTACH_REQ`.

mx_addr_length
Specifies the Physical Point of Attachment (PPA) to which to associate the *Style 2* Stream. This is a variable length identifier whose length is determined by the *mx_addr_length* value. Specifies the length of the Physical Point of Attachment (PPA) address. The form of the PPA address is provider-specific.

mx_addr_offset
Specifies the offset, from the beginning of the `M_PROTO` message block, of the start of the Physical Point of Attachment (PPA) address.

mx_flags Specifies the options flags for attachment. Options flags are provider-specific.

State

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

New State

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

Response

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

- **Successful**: The MXS provider acknowledges receipt of the primitive and successful outcome of the attach service with a `MX_OK_ACK` primitive. The new state is `MXS_ATTACHED`.

- **Unsuccessful (non-fatal errors)**: The MXS provider acknowledges receipt of the primitive and failure of the attach service with a `MX_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:

`[MXSYSERR]`	UNIX system error.
`[MXBADADDR]`	Bad address format or content.
`[MXOUTSTATE]`	Interface out of state.
`[MXBADOPT]`	Bad options format or content.
`[MXBADPARM]`	Bad parameter format or content.
`[MXBADPARMTYPE]`	Bad paramater structure type.
`[MXBADFLAG]`	Bad flag.
`[MXBADPRIM]`	Bad primitive.
`[MXNOTSUPP]`	Primitive not supported.
`[MXBADSLOT]`	Bad multplex slot.

4.1.3.2 MX_DETACH_REQ

Description

This MXS user originated primitive requests 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* MXS provider Streams, that is, Streams that return `MX_STYLE2` in the *mx_style* field of the `MX_INFO_ACK`.

Format

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

```
typedef struct MX_detach_req {
    mx_ulong mx_primitive;
} MX_detach_req_t;
```

Parameters

The detach request service primitive contains the following parameters:

mx_primitive
Specifies the service primitive type. Always `MX_DETACH_REQ`.

State

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

New State

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

Response

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

- **Successful**: The MXS provider acknowledges receipt of the primitive and successful outcome of the detach service with a `MX_OK_ACK` primitive. The new state is `MXS_DETACHED`.
- **Unsuccessful (non-fatal errors)**: The MXS provider acknowledges receipt of the primitive and failure of the detach service with a `MX_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:

[MXSYSERR]	UNIX system error.
[MXBADADDR]	Bad address format or content.
[MXOUTSTATE]	Interface out of state.
[MXBADOPT]	Bad options format or content.
[MXBADPARM]	Bad parameter format or content.
[MXBADPARMTYPE]	Bad paramater structure type.
[MXBADFLAG]	Bad flag.
[MXBADPRIM]	Bad primitive.
[MXNOTSUPP]	Primitive not supported.
[MXBADSLOT]	Bad multplex slot.

4.1.4 Initialization Service Primitives

Initialization service primitives allow the MXS 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 multiplex identifier as defined in G.703, it may be necessary to perform switching to connect or disconnect the circuit identification code associated with the multiplex identifier.

These service primitives implement the initialization service (see Section 3.1.4 [Initialization Service], page 18).

4.1.4.1 MX_ENABLE_REQ

Description

This MXS user originated primitive requests that the MXS 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 MX_enable_req {
    mx_ulong mx_primitive;
    mx_ulong mx_addr_length;
    mx_ulong mx_addr_offset;
    mx_ulong mx_flags;
} MX_enable_req_t;
```

Parameters

The enable request service primitive contains the following parameters:

mx_primitive
Specifies the service primitive type. Always `MX_ENABLE_REQ`.

mx_addr_length
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 value of this field. This remote address could be a circuit identification code, an IP address, or some other form of circuit or multiplex identifier.

mx_addr_offset
Specifies the offset, from the beginning of the `M_PROTO` message block, of the start of the remote address.

mx_flags Specifies the options flags associated with the enable request. Options flags are provider-specific.

State

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

New State

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

Response

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

- **Successful**: When successful, the MXS provider acknowledges successful completion of the enable service with a MX_ENABLE_CON primitive. The new state is MXS_ENABLED.
- **Unsuccessful (non-fatal errors)**: When unsuccessful, the MXS provider acknowledges the failure of the enable service with a MX_ERROR_ACK primitive containing the error. The new state remains unchanged.

Reasons for Failure

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

[MXSYSERR]	UNIX system error.
[MXBADADDR]	Bad address format or content.
[MXOUTSTATE]	Interface out of state.
[MXBADOPT]	Bad options format or content.
[MXBADPARM]	Bad parameter format or content.
[MXBADPARMTYPE]	Bad paramater structure type.
[MXBADFLAG]	Bad flag.
[MXBADPRIM]	Bad primitive.
[MXNOTSUPP]	Primitive not supported.
[MXBADSLOT]	Bad multplex slot.

4.1.4.2 MX_ENABLE_CON

Description

This MXS provider originated primitive is issued by the MXS 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 MX_enable_con {
    mx_ulong mx_primitive;
    mx_ulong mx_addr_length;
    mx_ulong mx_addr_offset;
    mx_ulong mx_flags;
} MX_enable_con_t;
```

Parameters

The enable confirmation service primitive contains the following parameters:

mx_primitive
Indicates the service primitive type. Always `MX_ENABLE_CON`.

mx_addr_length
Confirms the length of the remote address to which the enable is confirmed.

mx_addr_offset
Confirms the offset, from the beginning of the `M_PROTO` message block, of the start of the remote address.

mx_flags Confirms the options flags associated with the enable confirmation. Options flags are provider-specific.

State

This primitive is issued by the MXS provider in the `MXS_WCON_EREQ` state.

New State

The new state is `MXS_ENABLED`.

4.1.4.3 MX_DISABLE_REQ

Description

This MXS user originated primitive requests that the MXS 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 MX_disable_req {
    mx_ulong mx_primitive;
} MX_disable_req_t;
```

Parameters

The disable request service primitive contains the following parameters:

mx_primitive
Specifies the service primitive type. Always `MX_DISABLE_REQ`.

State

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

New State

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

Response

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

- **Successful**: When successful, the MXS provider acknowledges successful completion of the disable service with an `MX_DISABLE_CON` primitive. The new state is `MXS_ATTACHED`.
- **Unsuccessful (non-fatal errors)**: When unsuccessful, the MXS provider acknowledges the failure of the disable service with a `MX_ERROR_ACK` primitive containing the error. The new state remains unchanged.

Reasons for Failure

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

`[MXSYSERR]`	UNIX system error.
`[MXBADADDR]`	Bad address format or content.
`[MXOUTSTATE]`	Interface out of state.
`[MXBADOPT]`	Bad options format or content.
`[MXBADPARM]`	Bad parameter format or content.
`[MXBADPARMTYPE]`	Bad paramater structure type.
`[MXBADFLAG]`	Bad flag.
`[MXBADPRIM]`	Bad primitive.
`[MXNOTSUPP]`	Primitive not supported.
`[MXBADSLOT]`	Bad multplex slot.

4.1.4.4 MX_DISABLE_CON

Description

This MXS provider originated primitive is issued by the MXS 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 MX_disable_con {
    mx_ulong mx_primitive;
} MX_disable_con_t;
```

Parameters

The disable confirmation service primitive contains the following parameters:

mx_primitive
Indicates the service primitive type. Always `MX_DISABLE_CON`.

State

This primitive is issued by the MXS provider in the `MXS_WCON_RREQ` state.

New State

The new state is `MXS_ATTACHED`.

4.1.4.5 MX_DISABLE_IND

Description

This MXS provider originated primitive is issued by the MXS provider, if an autonomous event results in the disabling of the MXS use Stream without an explicity MXS user request.

Format

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

```
typedef struct MX_disable_ind {
    mx_ulong mx_primitive;
    mx_ulong mx_cause;
} MX_disable_ind_t;
```

Parameters

mx_primitive
Indicates the service primitive type. Always `MX_DISABLE_IND`.

mx_cause Indicates the cause of the autonomous disabling of the MXS user Stream.

State

This primitive will only be issued by the MXS provider in the `MXS_ENABLED` state.

New State

The new state is `MXS_ATTACHED`.

4.1.5 Options Management Service Primitives

The options management service primitives allow the MXS user to negotiate options with the MXS 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 19).

4.1.5.1 MX_OPTMGMT_REQ

Description

This MXS user originated primitive requests that MXS 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 MX_optmgmt_req {
    mx_ulong mx_primitive;
    mx_ulong mx_opt_length;
    mx_ulong mx_opt_offset;
    mx_ulong mx_mgmt_flags;
} MX_optmgmt_req_t;
```

Parameters

The option management request service primitive contains the following parameters:

mx_primitive
: Specifies the service primitive type. Always `MX_OPTMGMT_REQ`.

mx_opt_length
: Specifies the length of the options.

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

mx_mgmt_flags
: Specifies the management flags that determine what operation the MXS provider is expected to perform on the specified options. This field can assume one of the following values:

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

`MX_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 MXS provider.

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

`MX_CURRENT`
: Return the current value for the specified options (or all options). Do not alter the current value assumed by the MXS 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 MXS provider to acknowledge receipt of the primitive as follows:

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

Reasons for Failure

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

`[MXSYSERR]`	UNIX system error.
`[MXBADADDR]`	Bad address format or content.
`[MXOUTSTATE]`	Interface out of state.
`[MXBADOPT]`	Bad options format or content.
`[MXBADPARM]`	Bad parameter format or content.
`[MXBADPARMTYPE]`	Bad paramater structure type.
`[MXBADFLAG]`	Bad flag.
`[MXBADPRIM]`	Bad primitive.
`[MXNOTSUPP]`	Primitive not supported.
`[MXBADSLOT]`	Bad multplex slot.

4.1.5.2 MX_OPTMGMT_ACK

Description

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

Format

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

```
typedef struct MX_optmgmt_ack {
    mx_ulong mx_primitive;
    mx_ulong mx_opt_length;
    mx_ulong mx_opt_offset;
    mx_ulong mx_mgmt_flags;
} MX_optmgmt_ack_t;
```

Parameters

The option management acknowledgement service primitive contains the following parameters:

mx_primitive
: Indicates the service primitive type. Always `MX_OPTMGMT_ACK`.

mx_opt_length
: Indicates the length of the returned options.

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

mx_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:

`MX_SUCCESS`
: The MXS provider succeeded in negotiating or returning all of the options specified by the MXS user in the `MX_OPTMGMT_REQ` primitive.

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

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

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

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

State

This primitive is issued by the MXS provider in direct response to a `MX_OPTMGMT_REQ` primitive.

New State

The new state remains unchangted.

Rules

The MXS provider observes the following rules when processing option management service requests:

— When the *mx_mgmt_flags* field in the `MX_OPTMGMT_REQ` primitive is set to `MX_NEGOTIATE`, the MXS provider will attempt to negotiate a value for each of the options specified in the request.

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

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

— When the flags are `MX_CHECK`, the MXS 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 MXS provider to indicate asynchronous errors, events and statistics collection to the MXS user.

These service primitives implement the event reporting service (see Section 3.1.8 [Event Reporting Service], page 21).

4.1.6.1 MX_ERROR_IND

Description

This MXS provider originated service primitive is issued by the MXS 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 MX_error_ind {
    mx_ulong mx_primitive;
    mx_ulong mx_error_type;
    mx_ulong mx_unix_error;
    mx_ulong mx_state;
} MX_error_ind_t;
```

Parameters

The error indication service primitive contains the following parameters:

mx_primitive

Indicates the service primitive type. Always `MX_ERROR_IND`.

MX_error_type

Indicates the MXI error number describing the error. This field can have one of the following values:

`[MXSYSERR]`	UNIX system error.
`[MXBADADDR]`	Bad address format or content.
`[MXOUTSTATE]`	Interface out of state.
`[MXBADOPT]`	Bad options format or content.
`[MXBADPARM]`	Bad parameter format or content.
`[MXBADPARMTYPE]`	Bad paramater structure type.
`[MXBADFLAG]`	Bad flag.
`[MXBADPRIM]`	Bad primitive.
`[MXNOTSUPP]`	Primitive not supported.
`[MXBADSLOT]`	Bad multplex slot.

mx_unix_error

Indicates the reason for failure. This field is protocol-specific. When the *mx_error_type* field is `[MXSYSERR]`, the *mx_unix_error* field is the UNIX error number as described in `errno(3)`.

mx_state

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

MXS_UNINIT	Unitialized.
MXS_UNUSABLE	Device cannot be used, Stream in hung state.
MXS_DETACHED	No PPA attached, awaiting MX_ATTACH_REQ.
MXS_WACK_AREQ	Waiting for attach.
MXS_WACK_UREQ	Waiting for detach.
MXS_ATTACHED	PPA attached, awaiting MX_ENABLE_REQ.
MXS_WCON_EREQ	Waiting to send MX_ENABLE_CON.
MXS_WCON_RREQ	Waiting to send MX_DISABLE_CON.
MXS_ENABLED	Ready for use, awaiting primitive exchange.
MXS_WACK_CREQ	Waiting acknolwedgement of MX_CONNECT_REQ.
MXS_WCON_CREQ	Waiting to send MX_CONNECT_CON.
MXS_WACK_DREQ	Waiting acknolwedgement of MX_DISCONNECT_REQ.
MXS_WCON_DREQ	Waiting to send MX_DISCONNECT_CON.
MXS_CONNECTED	Connected, active data transfer.

State

This primitive can be issued in any state for which a local acknowledgement is not pending. The MXS 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 MX_STATS_IND

Description

This MXS provider originated primitive is issued by the MXS 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 MX_stats_ind {
    mx_ulong mx_primitive;
    mx_ulong mx_interval;
    mx_ulong mx_timestamp;
} MX_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:

mx_primitive
: Indicates the service primitive type. Always `MX_STATS_IND`.

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

mx_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 MXS provider in any state in which a local acknowledgement is not pending.

New State

The new state remains unchanged.

4.1.6.3 MX_EVENT_IND

Description

This MXS provider originated primitive is issued by the MXS 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 MX_event_ind {
    mx_ulong mx_primitive;
    mx_ulong mx_event;
    mx_ulong mx_slot;
} MX_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:

mx_primitive
Indicates the service primitive type. Always MX_EVENT_IND.

mx_event Indicates the provider-specific event that has occured.

MXF_EVT_DCD_ASSERT	Data carrier detect lead asserted.
MXF_EVT_DCD_DEASSERT	Data carrier detect lead deasserted.
MXF_EVT_DSR_ASSERT	Data set ready lead asserted.
MXF_EVT_DSR_DEASSERT	Data set ready lead deasserted.
MXF_EVT_DTR_ASSERT	Data terminal ready lead asserted.
MXF_EVT_DTR_DEASSERT	Data terminal ready lead deasserted.
MXF_EVT_RTS_ASSERT	Request to send lead asserted.
MXF_EVT_RTS_DEASSERT	Request to send lead deasserted.
MXF_EVT_CTS_ASSERT	Clear to send lead asserted.
MXF_EVT_CTS_DEASSERT	Clear to send lead deasserted.
MXF_EVT_RI_ASSERT	Ring indicator asserted.
MXF_EVT_RI_DEASSERT	Ring indicator deasserted.
MXF_EVT_YEL_ALARM	Yellow alarm condition.
MXF_EVT_BLU_ALARM	Blue alarm condition.
MXF_EVT_RED_ALARM	Red alarm condition.
MXF_EVT_NO_ALARM	Alarm recovery condition.

mx_slot Where the PPA is associated with a multiplexed medium, this parameter indicates the slots within the mutliplexed media to which the event corresponds. The form of the slot specification is provider- and media-specific. See also [Multiplex Media], page 12.

Where the PPA specifies a single channel for a medium, this parameter is set to zero ('0') by the MXS provider on MXS provider originated primitives and is ignored by the MXS provider on MXS user originated primitives.

State

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

New State

The new state remains unchanged.

4.2 Protocol Service Primitives

Protocol service primitives implement the Multiplex Interface protocol. Protocol service primitives provide the MXS user with the ability to connect transmission or reception directions of the bit stream, pass bits for transmission and accept received bits.

These service primitives implement the protocol services (see Section 3.2 [Protocol Services], page 21).

4.2.1 Connection Service Primitives

The connection service primitives permit the MXS user to establish a connection between the line (circuit or channel) and the MXS user in the transmit, receive, or both, directions.

These service primitives implement the connection service (see Section 3.2.1 [Connection Service], page 21).

4.2.1.1 MX_CONNECT_REQ

Description

This MXS user originated service primitive allows the MXS user to connect the user Stream to the medium in the transmit, receive, or both, directions.

Format

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

```
typedef struct MX_connect_req {
    mx_ulong mx_primitive;
    mx_ulong mx_conn_flags;
    mx_ulong mx_slot;
} MX_connect_req_t;
```

Parameters

The connect request service primitive contains the following parameters:

mx_primitive
: Specifies the service primitive type. Always `MX_CONNECT_REQ`.

mx_conn_flags
: Specifies the direction in which to connect. This field can contain a bitwise OR of one or more of the following flags:

`MXF_RX_DIR` Specifies that the MXS user Stream is to be connected to the medium in the receive direction.

`MXF_TX_DIR` Specifies that the MXS user Stream is to be connected to the medium in the transmit direction.

`MXF_MONITOR` Specifies that the MXS user Stream is to be connected to the medium in monitoring (tap) mode.

mx_slot Where the PPA is associated with a multiplexed medium, this parameter specifies the slots within the mutliplexed media to be connected to the MXS User Stream. The form of the slot specification is provider- and media-specific. See also [Multiplex Media], page 12.

Where the PPA specifies a single channel for a medium, this parameter is set to zero (‘0’) by the MXS provider on MXS provider originated primitives and is ignored by the MXS provider on MXS user originated primitives.

State

This service primitive is only valid in the MXS_ENABLED state.

New State

The new state is the MXS_WACK_CREQ state.

Response

The connect request service primitive requires that the MXS provider acknowledge receipt of the primitive as follows:

- **Successful**: When successful, the MXS provider acknolwedges successful completion of the connect service wtih a MX_OK_ACK primitive. The new state is MXS_WCON_CREQ. When the MXS provider eventually completes the connection, it confirms the connection with a MX_CONNECT_CON primitive and the new state is then MXS_CONNECTED.
- **Unsuccessful (non-fatal errors)**: When unsuccessful, the MXS provider acknowledges the failure of the connect service with a MX_ERROR_ACK primitive containing the error. The new state remains unchanged.

Reasons for Failure

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

[MXSYSERR]	UNIX system error.
[MXBADADDR]	Bad address format or content.
[MXOUTSTATE]	Interface out of state.
[MXBADOPT]	Bad options format or content.
[MXBADPARM]	Bad parameter format or content.
[MXBADPARMTYPE]	Bad paramater structure type.
[MXBADFLAG]	Bad flag.
[MXBADPRIM]	Bad primitive.
[MXNOTSUPP]	Primitive not supported.
[MXBADSLOT]	Bad multplex slot.

4.2.1.2 MX_CONNECT_CON

Description

This MXS provider originated service primitive allows the MXS provider to confirm the succesful completion of the connect servivce with the connection of the user Stream to the medium in the transmit, receive, or both, directions.

Format

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

```
typedef struct MX_connect_con {
    mx_ulong mx_primitive;
    mx_ulong mx_conn_flags;
    mx_ulong mx_slot;
} MX_connect_con_t;
```

Parameters

mx_primitive

Indicates the service primitive type. Always `MX_CONNECT_CON`.

mx_conn_flags

Indicates the connect flags. This field is a bitwise OR of zero or more of the following flags:

`MXF_RX_DIR` Confirms that the MXS user Stream was connected to the medium in the receive direction.

`MXF_TX_DIR` Confirms that the MXS user Stream was connected to the medium in the transmit direction.

`MXF_MONITOR` Confirms that the MXS user Stream was connected to the medium in monitoring (tap) mode.

mx_slot Where the PPA is associated with a multiplexed medium, this parameter specifies the slots within the mutliplexed media that are confirmed connected to the MXS user Stream. The form of the slot specification is provider- and media-specific. See also [Multiplex Media], page 12.

Where the PPA specifies a single channel for a medium, this parameter is set to zero (‘0’) by the MXS provider on MXS provider originated primitives and is ignored by the MXS provider on MXS user originated primitives.

State

This primitive will only be issued by the MXS provider in the `MXS_WCON_CREQ` state.

New State

The new state of the interface is the `MXS_CONNECTED` state.

4.2.2 Data Transfer Service Primitives

The data transfer service primitives permit the MXS user to pass bits for transmission to the MXS provider and accept received bits from the MXS provider.

These service primitives implement the data transfer service (see Section 3.2.2 [Data Transfer Service], page 22).

4.2.2.1 MX_DATA_REQ

Description

This MXS user originated primitive allows the MXS user to specify bits for transmission on the medium.

Format

The transmission request service primitive consists of one optional `M_PROTO` message block followed by one or more `M_DATA` message blocks containing the bits for transmission. The `M_PROTO` message block is structured as follows:

```
typedef struct MX_data_req {
    mx_ulong mx_primitive;
    mx_ulong mx_slot;
} MX_data_req_t;
```

Parameters

The transmission request service primitive contains the following parameters:

mx_primitive

Specifies the service primitive type. Always `MX_DATA_REQ`.

mx_slot Where the PPA is associated with a multiplexed medium, this parameter specifies the slots within the mutliplexed media upon which the user data is to be transmitted. The form of the slot specification is provider- and media-specific. See also [Multiplex Media], page 12.

Where the PPA specifies a single channel for a medium, this parameter is set to zero (‘`0`’) by the MXS provider on MXS provider originated primitives and is ignored by the MXS provider on MXS user originated primitives.

State

This primitive is only valid in the `MXS_CONNECTED` state.

New State

The state remains unchanged.

Response

Reasons for Failure

4.2.2.2 MX_DATA_IND

Description

This MXS provider originated primitive is issued by the MXS provider to indicate bits that were received on the medium.

Format

The receive indication service primitive consists of one optional `M_PROTO` message block followed by one or more `M_DATA` message blocks containing the received bits. The `M_PROTO` message block is structured as follows:

```
typedef struct MX_data_ind {
    mx_ulong mx_primitive;
    mx_ulong mx_slot;
} MX_data_ind_t;
```

Parameters

The receive indication service primitive contains the following parameters:

mx_primitive
Indicates the service primitive type. Always `MX_DATA_IND`.

mx_slot Where the PPA corresponds to a multiplexed media, this parameter specifies to which of the media streams the data indicated corresponds. The form of the slot specification is provider- and media-specific. See also [Multiplex Media], page 12.

Where the PPA specifies a single channel for a medium, this parameter is set to zero (‘`0`’) by the MXS provider on MXS provider originated primitives and is ignored by the MXS provider on MXS user originated primitives.

State

This primitive is only issued by the MXS provider in the `MXS_CONNECTED` state.

New State

The state remains unchanged.

Response

Reasons for Failure

4.2.3 Disconnection Service Primitives

The disconnection service primitives permit the MXS user to disconnect the Stream from the line (circuit or channel) for the transmit, receive, or both, directions. They also allow the MXS provider to indicate that a disconnection has occured outside of MXS user control.

These service primitives implement the disconnection service (see Section 3.2.3 [Disconnection Service], page 23).

4.2.3.1 MX_DISCONNECT_REQ

Description

This MXS user originated service primitive allows the MXS user to disconnect the MXS user Stream from the bit-stream in the transmit, receive, or both, directions.

Format

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

```
typedef struct MX_disconnect_req {
    mx_ulong mx_primitive;   /* always MX_DISCONNECT_REQ */
    mx_ulong mx_conn_flags;  /* direction to disconnect */
    mx_ulong mx_slot;        /* slot within multiplex */
} MX_disconnect_req_t;
```

Parameters

The disconnect request service primitive contains the following parameters:

mx_primitive
Specifies the service primitive type. Always `MX_DISCONNECT_REQ`.

mx_conn_flags
Specifies the direction from which to disconnect. This field can be a bitwise OR of one or more of the following flags:

`MXF_RX_DIR` Specifies that the MXS user Stream is to be disconnected from the medium in the receive direction.

`MXF_TX_DIR` Specifies that the MXS user Stream is to be disconnected from the medium in the transmit direction.

`MXF_MONITOR` Specifies that the MXS user Stream is to be discconnected from the medium in monitoring (tap) mode.

mx_slot Where the PPA is associated with a multiplexed medium, this parameter specifies the slots within the mutliplexed media that have been autonomouosly disconnected. The form of the slot specification is provider- and media-specific. See also [Multiplex Media], page 12.

Where the PPA specifies a single channel for a medium, this parameter is set to zero (‘0’) by the MXS provider on MXS provider originated primitives and is ignored by the MXS provider on MXS user originated primitives.

State

This service primitive is only valid in the `MXS_CONNECTED` state.

New State

The state remains unchanged.

Response

The disconnect request service primitive requires that the MXS provider acknowledge receipt of the primitive as follows:

- **Successful**: When successful, the MXS provider acknolwedges successful completion of the connect service wtih a `MX_OK_ACK` primitive. The new state is `MXS_WCON_DREQ`. When the MXS provider eventually completes the disconnection, it confirms the disconnect with a `MX_DISCONNECT_CON` primitive and the new state is then `MXS_ENABLED`.
- **Unsuccessful (non-fatal errors)**: When unsuccessful, the MXS provider acknowledges the failure of the connect service with a `MX_ERROR_ACK` primitive containing the error. The new state remains unchanged.

Reasons for Failure

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

`[MXSYSERR]`	UNIX system error.
`[MXBADADDR]`	Bad address format or content.
`[MXOUTSTATE]`	Interface out of state.
`[MXBADOPT]`	Bad options format or content.
`[MXBADPARM]`	Bad parameter format or content.
`[MXBADPARMTYPE]`	Bad paramater structure type.
`[MXBADFLAG]`	Bad flag.
`[MXBADPRIM]`	Bad primitive.
`[MXNOTSUPP]`	Primitive not supported.
`[MXBADSLOT]`	Bad multplex slot.

4.2.3.2 MX_DISCONNECT_CON

Description

This MXS provider originated primitive is issued by the MXS provider to confirm the successful completion of the disconnect service with the disconnection of the user Stream from the medium in the transmit, receive, or both, directions.

Format

```
typedef struct MX_disconnect_con {
    mx_ulong mx_primitive;
    mx_ulong mx_conn_flags;
    mx_ulong mx_slot;
} MX_disconnect_con_t;
```

Parameters

mx_primitive
Indicates the service primitive type. Always `MX_DISCONNECT_CON`.

mx_conn_flags
Indicates the connect flags. This field is a bitwise OR of zero or more of the following flags:

`MXF_RX_DIR` Confirms that the MXS user Stream was disconnected from the medium in the receive direction.

`MXF_TX_DIR` Confirms that the MXS user Stream was disconnected from the medium in the transmit direction.

`MXF_MONITOR` Confirms that the MXS user Stream was discconnected from the medium in monitoring (tap) mode.

mx_slot Where the PPA is associated with a multiplexed medium, this parameter indicates the slots within the mutliplexed media that are confirmed as disconnected. The form of the slot specification is provider- and media-specific. See also [Multiplex Media], page 12.

Where the PPA specifies a single channel for a medium, this parameter is set to zero (‘`0`’) by the MXS provider on MXS provider originated primitives and is ignored by the MXS provider on MXS user originated primitives.

State

This primitive will only be issued by the MXS provider in the `MXS_WCON_DREQ` state.

New State

The new state of the interface is the `MXS_ENABLED` state.

4.2.3.3 MX_DISCONNECT_IND

Description

This MXS provider originated primitive is issued by the MXS provider if an autonomous event results in the disconnection of the transmit and receive bit-streams from the MXS user without an explicit MXS user request.

Format

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

```
typedef struct MX_disconnect_ind {
    mx_ulong mx_primitive;   /* always MX_DISCONNECT_IND */
    mx_ulong mx_conn_flags;  /* direction disconnected */
    mx_ulong mx_cause;       /* cause for disconnection */
    mx_ulong mx_slot;        /* slot within multiplex */
} MX_disconnect_ind_t;
```

Parameters

mx_primitive
Indicates the service primitive type. Always `MX_DISCONNECT_IND`.

mx_conn_flags
Indicates the connect flags. This field is a bitwise OR of zero or more of the following flags:

`MXF_RX_DIR` Indicates that the MXS user Stream disconnected from the medium in the receive direction.

`MXF_TX_DIR` Indicates that the MXS user Stream disconnected from the medium in the transmit direction.

`MXF_MONITOR` Indicates that the MXS user Stream discconnected from the medium in monitoring (tap) mode.

mx_cause Indicates the cause of the autonomous disconnect.

mx_slot Where the PPA is associated with a multiplexed medium, this parameter indicates the slots within the mutliplexed media that have autonomously disconnected. The form of the slot specification is provider- and media-specific. See also [Multiplex Media], page 12.

Where the PPA specifies a single channel for a medium, this parameter is set to zero (‘`0`’) by the MXS provider on MXS provider originated primitives and is ignored by the MXS provider on MXS user originated primitives.

State

This primtiive will only be issued by the MXS provider in the `MXS_CONNECTED` state.

New State

The new state is `MXS_ENABLED`.

4.3 Diagnostics Requirements

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

4.3.1 Non-Fatal Error Handling Facility

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

4.3.2 Fatal Error Handling Facility

These errors are issued by the MX provider when it detects errors that are not correctable by the MX user, or if it is unable to report a correctible error to the MX 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 MXS 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.

5 MXI Input-Output Controls

These input-output controls can be used to interrogate, negotiate, reset, collect and manage a given channel or group of channels. When issued on a MXS user Stream, they can only be used to affect the channel or channels associated with the MXS user Stream. Deattached *Style 2* Streams have no associated channels. When issued on a management Stream, they can be used to affect the configuration of any channel or channels accessible to the management Stream (i.e. provided by the same driver, or temporarily linked from the control Stream).

Channels can have characteristics at the channel level, as well as characteristics at the channel group level. For example, the channel may not be looped back at the channel, but might be looped back at the channel group (span). Where the channel represents a channel within a multiplexed medium (such as a PCM TDM facility), the MXI input-output controls can be used to interrogate, negotiate and otherwise manage the channel group characteristics providing that the MXS user has sufficient privilege to do so.

Note that these input-output controls are not normally issued on the global management Stream by user processes. Rather the Management Agent (SNMP Agent) for the driver is normally responsible for managing channels within the driver using these input-output controls. Nomally these input-output controls would only be issued by user processes to affect the channel or channels associated with the attached MXS user Stream.

5.1 MXI Configuration

These input-output controls can be used to interrogate or negotiate the configuration of a given channel or group of channels.

```
typedef struct mx_config {
    mx_ulong type;         /* unused */
    mx_ulong encoding;     /* encoding */
    mx_ulong block_size;   /* data block size (bits) */
    mx_ulong samples;      /* samples per block */
    mx_ulong sample_size;  /* sample size (bits) */
    mx_ulong rate;         /* clock rate (samples/second) */
    mx_ulong tx_channels;  /* number of tx channels */
    mx_ulong rx_channels;  /* number of rx channels */
    mx_ulong opt_flags;    /* options flags */
} mx_config_t;
```

The multiplex configuration structure, `mx_config_t`, contains the following members:

type This member is only to maintain alignment with the equivalient parameter structure as defined in the MXI and unused in the input-output control.

encoding Indicates or specifies the encoding associated with the multiplex. When the multiplex is used for any form of data, `MX_ENCODING_NONE` will be indicated and should be specified. *encoding* can be one of the following values:

`MX_ENCODING_NONE`	No encoding. Used for data or other clear channel information.
`MX_ENCODING_CN`	CN.
`MX_ENCODING_DVI4`	DVI4.
`MX_ENCODING_FS1015`	FIPS FS 1015 LPC.
`MX_ENCODING_FS1016`	FIPS FS 1016 LPC.
`MX_ENCODING_G711_PCM_A`	G.711 PCM A-law.

`MX_ENCODING_G711_PCM_L`	G.711 PCM Linear.
`MX_ENCODING_G711_PCM_U`	G.711 PCM Mu-law.
`MX_ENCODING_G721`	G.721.
`MX_ENCODING_G722`	G.722.
`MX_ENCODING_G723`	G.723.
`MX_ENCODING_G726`	G.726.
`MX_ENCODING_G728`	G.728.
`MX_ENCODING_G729`	G.729.
`MX_ENCODING_GSM`	GSM.
`MX_ENCODING_GSM_EFR`	GSM Extended Full-Rate.
`MX_ENCODING_GSM_HR`	GSM Half-Rate.
`MX_ENCODING_LPC`	LPC.
`MX_ENCODING_MPA`	MPA.
`MX_ENCODING_QCELP`	QCELP.
`MX_ENCODING_RED`	RED.
`MX_ENCODING_S16_BE`	Signed 16-bit Big-Endian.
`MX_ENCODING_S16_LE`	Signed 16-bit Little-Endian.
`MX_ENCODING_S8`	Sign 8-bit.
`MX_ENCODING_U16_BE`	Unsigned 16-bit Big-Endian.
`MX_ENCODING_U16_LE`	Unsigned 16-bit Little-Endian.
`MX_ENCODING_U8`	Unsigned 8-bit.
`MX_ENCODING_VDVI`	DVI.

block_size Specifies or indicates the block size associated with the multiplex. The block size is the number of samples that are written or read at one time. If this value is less than the size of a STREAMS fast buffer, `FASTBUF`, then a `FASTBUF` of samples will be read or written at once.

samples Specifies or indicates the number of samples (from the same timeslot) in a block.

sample_size

Specifies or indicates the sample size in bits. This can normally be 3, 4, 5, 7, 8, 12, 14 or 16.

rate Specifies or indicates the rate of the multiplex. This is the rate in samples per second. *rate* can be one of the following values:

`MX_RATE_VARIABLE`	The rate is variable.
`MX_RATE_8000`	56kbps or 64kbps.
`MX_RATE_11025`	11kHz Audio.
`MX_RATE_16000`	16kHz Audio.
`MX_RATE_22050`	22kHz Audio.
`MX_RATE_44100`	44kHz Audio.
`MX_RATE_90000`	90kHz Audio.
`MX_RATE_184000`	23B.
`MX_RATE_192000`	T1 (24B).
`MX_RATE_240000`	30B.
`MX_RATE_248000`	E1 (31B).

tx_channels

Specifies or indicates the number of transmit channels available. For the MX interface, this value is either 0 or 1.

rx_channels

Specifies or indicates the number of receive channels available. For the MX interface, this value is either 0, 1, or 2. (The value of 2 is used for monitoring mode where two receive channels exists and zero transmit channels.)

opt_flags Specifies or indicates the options associated with the MX provider. MX provider options are provider specific and no generic options have yet been defined.

5.1.1 MXI Get Configuration

`MX_IOCGCONFIG`

Gets the multiplex configuration. Upon success, the multiplex configuration is written to the memory extent indicated by the pointer argument to the `ioctl(2s)` call.

5.1.2 MXI Set Configuration

`MX_IOCSCONFIG`

Set the multiplex configuration. Upon success, the multiplex configuration is read from the memory extent specified by the pointer argument to the `ioctl(2s)` call.

5.1.3 MXI Test Configuration

`MX_IOCTCONFIG`

Test the multiplex configuration. Upon success, the multiplex configuration is read from the memory extent specified by the pointer argument to the `ioctl(2s)` call, values adjusted according to the rules for configuration, and the resulting configuraiton written back to the memory extent specified by the pointer argumnet to the `ioctl(2s)` call. Actual configuration is not changed.

5.1.4 MXI Commit Configuration

`MX_IOCCCONFIG`

Confirms the multiplex configuration. Upon success, the multiplex configuration is read from the memory extent specified by the pointer argument to the `ioctl(2s)` call, values adjusted according to the rules for configuration, the configuration applied, and then the resulting configuration written back to the memory extent specified by the pointer argument to the `ioctl(2s)` call.

Normally, the argument to the `MX_IOCCCONFIG` call is the same as to an immediately preceding `MX_IOCTCONFIG` call.

5.2 MXI Options

These input-output controls can used to interrogate or negotiate the options associated with a given channel or group of channels.

5.3 MXI State

These input-output controls can be used to interrotate or reset the state associated with a channel or a group of channels.

State input-output controls all take an argument containing a poitner to a `mx_statem_t` structure, formatted as follows:

```
typedef struct mx_statem {
    mx_ulong index;
    mx_ulong type;
    mx_ulong rate;
    mx_ulong mode;
    mx_ulong admin_state;
    mx_ulong usage_state;
    mx_ulong avail_status;
    mx_ulong ctrl_status;
} mx_statem_t;
```

The multiplex state structure, `mx_statem_t`, contains the following members:

index Provides time slot index for the channel. For T1 and J1 spans, the time slots '`1`' through '`24`' index the corresponding time slot in the span. For E1 spans, the time slot indices '`1`' throught '`31`' index the corresponding time slot in the span. For E1 operation, TS0 is unusable. For E1 CAS operation (where any channel in the span is configured for CAS), TS16 is not available to users for payload. For V.35 and other discrete synchronous channels, this index is '`1`'.

type Specifies or indicates whether the channel (or channels) has channel associated signalling or common channel signalling. This field can have one of the following values:

`MX_TYPE_NONE`
For non-trunk channels, no type is necessary.

`MX_TYPE_CAS`
For T1 and J1 span, channel associated signalling implies 56kbps DS0A operation for data within the channel.

`MX_TYPE_CCS`
For E1, T1 or J1 spans, common channel signalling implies 64kbps DS0 oepration within the channel is indicated. For E1, CCS operation for the entire span implies that channel 17 (timeslot 16) is used for common channel signalling or is also available for payload. This is why it is typical on non-CAS E1 spans to place the signalling channel in timeslot 16 (e.g. the D-channel of a primary rate interface).

rate Specifies or indicates the bit rate of the channel in a single-rate channel, or of each channel in a multi-rate channel, or of each channel in a full-rate channel. Channels '`1`' through '`24`' for T1 and J1 can be 56kbps or 64kbps. Channels '`1`' through '`31`' for E1 are 64kbps but can be forced into 56kbps mode. The default is 64kbps for E1 CCS and CAS channels and T1 CCS channels; 56kbps for T1 CAS channels.

mode Specifies or indicates the channel mode. This is bitwise OR of zero or more of the following values:

`MX_MODE_REMLOOP`
The receive data in the channel is looped back to replace the transmit data for the channel. This may either be accomplished within the host or using the per-channel loopback capability of some chip sets.

`MX_MODE_LOCLOOP`
The transmit data for the channel is looped back to replace the receive data for the channel. This may be accomplished within the host.

`MX_MODE_TEST`

The channel is marked for BERT testing. When BERT testing for the span is enabled on a channel basis, this channel will be included in the channels upon which the BERT test pattern is transmitted.

Because tests are disruptive, no value can be added to this set unless the channel has a control status of “subject to test” or “reserved for test”.

admin_state

Specifies or indicates the administrative state of the channel. The administrative state can be one of the following values:

`MX_ADMIN_LOCKED`

The administrative state is “locked”. The channel is administratively prohibited from providing service to users.

`MX_ADMIN_UNLOCKED`

The administrative state is “unlocked”. The channel is administratively permitted to provide service to users.

`MX_ADMIN_SHUTDOWN`

The administrative state is “shutting down”. The channel will continue to provide service to existing users but will reject new users: once there are no more users of the channel, the channel will move to the “locked” state.

usage_state Specifies or indicates the usage state of the channel. The usage state can be one of the following values:

`MX_USAGE_IDLE`

The channel is “idle”. The channel is not currently in use.

`MX_USAGE_ACTIVE`

The channel is “active”. The channel is in use and has sufficient operating capacity to provide for additional users simultaneously (e.g. a half-channel is used).

`MX_USAGE_BUSY`

The channel is “busy”. The channel is in use and has no spare capacity (i.e. the full channs is in use).

If partial channels are not supported, only the values “idle” and “busy” are allowed.

avail_status

Specifies or indicates the availabiltiy status of the channel. The availablity status is a bitwise OR of zero or more of the following values:

`MX_AVAIL_INTEST`

The channel is “in test”. The channel is undergoing a test procedure. The administrative state is “locked” and the operational state is “disabled”. This condition exists while the span is in test in a manner disruptive to the channel, or when the channel is in loopback or test modes.

`MX_AVAIL_FAILED`

The channel has “failed”. The channel has an internal fault that prevents it from operating. The operational state is “disabled”. This value is present when the same value is present in the span availability status.

`MX_AVAIL_POWEROFF`
: The channel has “power off”. The channl requires power to be applied and is not powered on. For example, power management may have removed power from the device. This value is present when the same value is present in the span availablity status.

`MX_AVAIL_OFFLINE`
: The channel is “off line”. The channel requires a outing operation to be performed to place it online and make it available for use. The operation may be manul or automatic, or both. The operational state is “disabled”. This value is present when the same value is present in the span availability status.

`MX_AVAIL_OFFDUTY`
: The channel is “off duty”. The channel has been made inactive by an internal control process in accordance with a predetermined time schedule. Under normal conditions, the control process can be expected to reactivate the channel at some scheduled time.

`MX_AVAIL_DEPEND`
: The channel has a “dependency”. The channel cannot operate because some other resource on which it depends is unavailable (e.g. the span).

`MX_AVAIL_DEGRADED`
: The channel is “degraded”. The channel is operating with degraded peformance. This value is present when the same value is present in the span availability status.

`MX_AVAIL_MISSING`
: The channel is “not installed”. The channel is not present in the system or is incomplete.

`MX_AVAIL_LOGFULL`
: Not used.

ctrl_status Specifies or indicates the control status of the channel. The control status is a bitwise OR of zero or more of the following values:

`MX_CTRL_CANTEST`
: The channel is “subject to test”. The channel is available to normal users but tests may be conducted on it simultaneously at unpredicatable times, which may cause it to exhibit unusual characteristics to users.

`MX_CTRL_PARTLOCK`
: The channel is “part of services locked”. A manager has adminstratively locked some part of the channel.

`MX_CTRL_RESERVED`
: The channel is “reserved for test”. The channel is undergoing a test procedure and is unavailable to users.

`MX_CTRL_SUSPENDED`
: The channel is “suspended”. The channel service has been administratively suspended to users.

5.3.1 MXI Get State

`MX_IOCGSTATEM`

Requests that the state information be obtained and written to the `mx_statem_t` structure pointed to by the argument to the input-output control.

5.3.2 MXI Reset State

`MX_IOCCMRESET`

Request that the state associated with the multiplex be reset. This input-output control takes no argument.

5.4 MXI Statistics

These input-output controls can be used to collect statistics or set staticstics collection intervals associated with a channel or group of channels.

Statistics input-output controls all take an argument containing a pointer to a `mx_stats_t` structure, formatted as follows:

```
typedef struct mx_stats {
    mx_ulong header;
    mx_ulong rx_octets;
    mx_ulong tx_octets;
    mx_ulong rx_overruns;
    mx_ulong tx_underruns;
    mx_ulong rx_buffer_overflows;
    mx_ulong tx_buffer_overflows;
    mx_ulong lead_cts_lost;
    mx_ulong lead_dcd_lost;
    mx_ulong carrier_lost;
    mx_ulong errored_seconds;
    mx_ulong severely_errored_seconds;
    mx_ulong severely_errored_framing_seconds;
    mx_ulong unavailable_seconds;
    mx_ulong controlled_slip_seconds;
    mx_ulong path_coding_violations;
    mx_ulong line_errored_seconds;
    mx_ulong bursty_errored_seconds;
    mx_ulong degraded_minutes;
    mx_ulong line_coding_violations;
} mx_stats_t;
```

The multiplex statistics structure, `mx_stats_t`, contains the following members:

header Specifies or indicates the statistics period header associated with the multiplex. This header is a statistics collection period in milliseconds.

rx_octets Indicates the number of octets received during the collection interval. This does not include octets for which there was a receiver overrun condition.

tx_octets Indicates the number of octets transmitted during the collection interval. This does not include octets for which there was a transmitter underrun condition.

rx_overruns

Indicates the number of receive overrun conditions that occurred during the collection interval. When the overrun condition spans interval boundaries, the condition is counted in the interval during which the overrun condition began.

tx_underruns

Indicates the number of transmitter underrun conditions that occurred during the collection interval. When the underrun condition spans interval boundaries, the condition is counted in the interval during which the underrun condition began.

rx_buffer_overflows

Indicates the number of receive buffer overflows that occured during the collection interval. Receive buffer overflow conditions occur when the driver is unable to allocate a message block or buffer for received bits, resulting in the discard of the received bits.

tx_buffer_overflows

Indicates the number of transmit buffer overflows that occured during the collection interval. Transmit buffer overflow conditions occur when the driver is unable to allocate a message block or buffer for transmit bits, resulting in the discard of the bits to be transmitted.

lead_cts_lost

Indicates the number of Clear To Send leads lost. That is, the number of times that the Clear To Send lead transitioned from asserted to deasserted.

lead_dcd_lost

Indicates the number of Data Carrier Detect leads lost. That is, the number of times that the Data Carrier Detect lead trasitioned from asserted to deasserted.

carrier_lost

Indicates the number of Carrier lost conditions. That is, the number of times that an alarm or lead indicated that the facility carrier was lost.

errored_seconds

The number of errored seconds (ESs) in the current interval. An errored second has one or more path code violations, one or more out of frame defects, one or more controlled slip events, or a detected alarm indication signal (AIS) defect.

severely_errored_seconds

The number of severely errored seconds (SESs) in the current interval.

severely_errored_framing_seconds

The number of severely errored framing seconds (SEFSs) in the current interval. A severely errored framing second has one or more out of frame defects or a detected AIS defect.

unavailable_seconds

The number of unavailable seconds in the current interval.

controlled_slip_seconds

The number of controlled slip seconds (CSSs) in the current interval. A controlled slip second has one or more controlled slip events.

path_coding_violations

The number of path coding violations (PCVs) in the current interval. A path coding violation is a fram synchronization bit error in the D4 and E1 no-CRC4 formats, or a CRC or frame synchronization bit error in the ESF and E1 CRC4 formats.

line_errored_seconds
The number of line errored seconds (LESs) in the current interval. A line errored second is a second in which one or more line code violation error events are detected.

bursty_errored_seconds
The number of bursty errroed seconds (BESs) in the current interval. A bursty errored second has 2 to 319 path coding violation error events, no severely errored frame defects, and no detected inocming AIS defects.

degraded_minutes
The number of degraded minutes (DMs) in the current interval.

line_coding_violations
The number of line coding violations (LCVs) in the current interval. An LCV is the occurence of a bipolar violation (BPV) or excessive zeroes (EXZ) error event.

5.5 MXI Events

These input-output controls can be used to specify the events that will be reported by a channel or channels.

Notification input-output controls all take an argument containing a pointer to a `mx_notify_t` structure, formatted as follows:

```
typedef struct mx_notify {
    mx_ulong events;
} mx_notify_t;
```

The multiplex events structure, `mx_notify_t`, contains the following members:

events Specifies or indicates a bitwise OR of the events associated wtih the multiplex. When a bit is set, it specifies that event reporting for the specific event is enabled for the multiplex; when clear, that the event reporting is disabled.

5.5.1 MXI Get Notify

`MX_IOCGNOTIFY`

Requests that the events associated with the multiplex be obtained and written to the `mx_notify_t` structure pointed to by the argument to the input-output control.

5.5.2 MXI Set Notify

`MX_IOCSNOTIFY`

Requests that the events associated with the multiplex be read from the `mx_notify_t` structure pointed to by the argument to the input-output control and set for the multiplex. Each bit set in the *events* member specifies an event for which notification is to be set.

5.5.3 MXI Clear Notify

`MX_IOCCNOTIFY`

Request that the events associated with the multiplex be read from the `mx_notify_t` structure pointed to by the argument to the input-output control and cleared for the multiplex. Each bit set in the *events* member specifies an event for which notification is to be cleared.

5.6 MXI Commands

These input-output controls can be used to manage a channel or channels.

Management input-output controls all take an argument containing a pointer to a `mx_mgmt_t` structure, formatted as follows:

```
typedef struct mx_mgmt {
    mx_ulong cmd;
} mx_mgmt_t;
```

The multiplex management structure, `mx_mgmt_t`, contains the following members:

cmd Specifies the management command to be performed by the MXS provider. This member can have one of the following values:

`MX_CMD_REMLOOP`
Place the multiplex in remote loopback. The administrative state of the multiplex must be "locked" for this command to be successfull. Once complete, the control status of the multiplex will contain "reserved for test" and the availability status of the multiplex will contain "in test".

`MX_CMD_LOCLOOP`
Place the multiplex in local loopback. The administrative state of the multiplex must be "locked" for this command to be successfull. Once complete, the control status of the multiplex will contain "reserved for test" and the availabiltiy status of the multiplex will contain "in test".

`MX_CMD_FORTEST`
Reserve the multiplex for BERT testing. The administrative state of the multiplex must be "locked" for this command to be successful. Once complete, the control status of the multiplex will contain "reserved for test" and the availability status of the multiplex will contain "in test" while BERT testing is actively being performed.

`MX_CMD_LOCK`
Place the multiplex in the "locked" administrative state. If the multiplex is in the "unlocked" or "shutting down" states and the usage state is "busy", this will result in the removal from service of the multiplex while it is in use.

`MX_CMD_UNLOCK`
Place the multiplex in the "unlocked" administrative state. This makes the multiplex adminstratively available for use.

`MX_CMD_SHUTDOWN`
Place the multiplex in the "shutting down" administrative state. If the multiplex has a usage state of "idle" the multiplex will be placed immediately into the "locked" administrative state. If the usage state is "busy", then the administrative state will be set to "shutting down" and the driver will wait until the multiplex is released before it is placed in the "locked" administrative state.

5.6.1 MXI Command

`MX_IOCCMGMT`

Request that the management command be read from the `mx_mgmt_t` structure pointed to by the argument to the input-output control and acted upon for the multiplex.

6 MXI Management

Appendix A MXI Header Files

A.1 MXI Header File Listing

```
#ifndef __SS7_MXI_H__
#define __SS7_MXI_H__

typedef int32_t mx_long;
typedef uint32_t mx_ulong;
typedef uint16_t mx_ushort;
typedef uint8_t mx_uchar;

#define MX_INFO_REQ              1U
#define MX_OPTMGMT_REQ           2U
#define MX_ATTACH_REQ            3U
#define MX_ENABLE_REQ            4U
#define MX_CONNECT_REQ           5U
#define MX_DATA_REQ              6U
#define MX_DISCONNECT_REQ        7U
#define MX_DISABLE_REQ           8U
#define MX_DETACH_REQ            9U

#define MX_INFO_ACK             10U
#define MX_OPTMGMT_ACK          11U
#define MX_OK_ACK               12U
#define MX_ERROR_ACK            13U
#define MX_ENABLE_CON           14U
#define MX_CONNECT_CON          15U
#define MX_DATA_IND             16U
#define MX_DISCONNECT_IND       17U
#define MX_DISCONNECT_CON       18U
#define MX_DISABLE_IND          19U
#define MX_DISABLE_CON          20U
#define MX_EVENT_IND            21U

/*
 *  MX STATES
 */
#define MXS_UNINIT              -2U
#define MXS_UNUSABLE            -1U
#define MXS_DETACHED             0U
#define MXS_WACK_AREQ            1U
#define MXS_WACK_UREQ            2U
#define MXS_ATTACHED             3U
#define MXS_WACK_EREQ            4U
#define MXS_WCON_EREQ            5U
#define MXS_WACK_RREQ            6U
#define MXS_WCON_RREQ            7U
#define MXS_ENABLED              8U
#define MXS_WACK_CREQ            9U
#define MXS_WCON_CREQ           10U
#define MXS_WACK_DREQ           11U
#define MXS_WCON_DREQ           12U
#define MXS_CONNECTED           13U
```

```
/*
 *  MX STATE FLAGS
 */
#define MXSF_UNINIT             (1<<(2+MXS_UNINIT))
#define MXSF_UNUSABLE           (1<<(2+MXS_UNUSABLE))
#define MXSF_DETACHED           (1<<(2+MXS_DETACHED))
#define MXSF_WACK_AREQ          (1<<(2+MXS_WACK_AREQ))
#define MXSF_WACK_UREQ          (1<<(2+MXS_WACK_UREQ))
#define MXSF_ATTACHED           (1<<(2+MXS_ATTACHED))
#define MXSF_WACK_EREQ          (1<<(2+MXS_WACK_EREQ))
#define MXSF_WCON_EREQ          (1<<(2+MXS_WCON_EREQ))
#define MXSF_WACK_RREQ          (1<<(2+MXS_WACK_RREQ))
#define MXSF_WCON_RREQ          (1<<(2+MXS_WCON_RREQ))
#define MXSF_ENABLED            (1<<(2+MXS_ENABLED))
#define MXSF_WACK_CREQ          (1<<(2+MXS_WACK_CREQ)
#define MXSF_WCON_CREQ          (1<<(2+MXS_WCON_CREQ))
#define MXSF_WACK_DREQ          (1<<(2+MXS_WACK_DREQ))
#define MXSF_WCON_DREQ          (1<<(2+MXS_WCON_DREQ))
#define MXSF_CONNECTED          (1<<(2+MXS_CONNECTED))

/*
 *  MX PROTOCOL PRIMITIVES
 */

/*
 *  MX_INFO_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_info_req {
        mx_ulong mx_primitive;          /* always MX_INFO_REQ */
} MX_info_req_t;

/*
 *  MX_INFO_ACK
 *  -------------------------------------------------------------------------
 *  Indicates to the multiplex user requested information concerning the
 *  multiplex provider and the attached multiplex (if any).
 */
typedef struct MX_info_ack {
        mx_ulong mx_primitive;          /* always MX_INFO_ACK */
        mx_ulong mx_addr_length;        /* multiplex address length */
        mx_ulong mx_addr_offset;        /* multiplex address offset */
        mx_ulong mx_parm_length;        /* multiplex paramters length */
        mx_ulong mx_parm_offset;        /* multiplex paramters offset */
        mx_ulong mx_prov_flags;         /* provider options flags */
        mx_ulong mx_prov_class;         /* provider class */
        mx_ulong mx_style;              /* provider style */
        mx_ulong mx_version;            /* multiplex interface version */
        mx_ulong mx_state;              /* multiplex state */
} MX_info_ack_t;

#define MX_CIRCUIT      0x01    /* circuit provider class */

#define MX_STYLE1       0x0     /* does not perform attach */
#define MX_STYLE2       0x1     /* does perform attach */
```

```
#define MX_VERSION_1_0  0x10    /* version 1.0 of interface */
#define MX_VERSION_1_1  0x11    /* version 1.1 of interface */
#define MX_VERSION      MX_VERSION_1_1

#define MX_PARMS_CIRCUIT        0x01    /* parms structure type */
typedef struct MX_parms_circuit {
        mx_ulong mp_type;               /* always MX_PARMS_CIRCUIT */
        mx_ulong mp_encoding;           /* encoding */
        mx_ulong mp_block_size;         /* data block size (bits) */
        mx_ulong mp_samples;            /* samples per block */
        mx_ulong mp_sample_size;        /* sample size (bits) */
        mx_ulong mp_rate;               /* channel clock rate (samples/second) */
        mx_ulong mp_tx_channels;        /* number of tx channels */
        mx_ulong mp_rx_channels;        /* number of rx channels */
        mx_ulong mp_opt_flags;          /* options flags */
} MX_parms_circuit_t;

#define MX_PARMS_CHANMAP        0x02    /* parms structure type */
typedef struct MX_parms_chanmap {
        mx_ulong mp_type;               /* always MX_PARM_CHANMAP */
        mx_ulong mp_spans;              /* number of spans */
        mx_ulong mp_span_offset;        /* offset of first span */
        mx_long mp_span_increment;      /* increment of next span from previous span */
        mx_ulong mp_slot_offset;        /* offset from beginning of span */
        mx_long mp_slot_increment;      /* increment of next slot from previous slot */
        mx_ulong mp_chan_map;           /* channel (bit) map (lsb = slot 0, msb = slot
                                           31) */
} MX_parms_chanmap_t;

union MX_parms {
        mx_ulong mp_type;               /* structure type */
        MX_parms_circuit_t circuit;     /* circuit structure */
        MX_parms_chanmap_t chanmap;     /* chanmap structure */
};

#define MX_PARM_OPT_CLRCH       0x01    /* supports clear channel */

#define MX_ENCODING_NONE         0
#define MX_ENCODING_CN           1
#define MX_ENCODING_DVI4         2
#define MX_ENCODING_FS1015       3
#define MX_ENCODING_FS1016       4
#define MX_ENCODING_G711_PCM_A   5
#define MX_ENCODING_G711_PCM_L   6
#define MX_ENCODING_G711_PCM_U   7
#define MX_ENCODING_G721         8
#define MX_ENCODING_G722         9
#define MX_ENCODING_G723        10
#define MX_ENCODING_G726        11
#define MX_ENCODING_G728        12
#define MX_ENCODING_G729        13
#define MX_ENCODING_GSM         14
#define MX_ENCODING_GSM_EFR     15
#define MX_ENCODING_GSM_HR      16
#define MX_ENCODING_LPC         17
```

```
#define MX_ENCODING_MPA         18
#define MX_ENCODING_QCELP       19
#define MX_ENCODING_RED         20
#define MX_ENCODING_S16_BE      21
#define MX_ENCODING_S16_LE      22
#define MX_ENCODING_S8          23
#define MX_ENCODING_U16_BE      24
#define MX_ENCODING_U16_LE      25
#define MX_ENCODING_U8          26
#define MX_ENCODING_VDVI        27

#define MX_RATE_VARIABLE        0
#define MX_RATE_8000            8000
#define MX_RATE_11025           11025
#define MX_RATE_16000           16000
#define MX_RATE_22050           22050
#define MX_RATE_44100           44100
#define MX_RATE_90000           90000
#define MX_RATE_184000          184000  /* 23B */
#define MX_RATE_192000          192000  /* T1 */
#define MX_RATE_240000          240000  /* 30B */
#define MX_RATE_248000          248000  /* E1 */
#define MX_RATE_5376000         5376000 /* T3 */

/*
 *  MX_OPTMGMT_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_optmgmt_req {
        mx_ulong mx_primitive;          /* always MX_OPTMGMT_REQ */
        mx_ulong mx_opt_length;         /* length of options */
        mx_ulong mx_opt_offset;         /* offset of options */
        mx_ulong mx_mgmt_flags;         /* option flags */
} MX_optmgmt_req_t;

/*
 *  MX_OPTMGMT_ACK
 *  -------------------------------------------------------------------------
 */
typedef struct MX_optmgmt_ack {
        mx_ulong mx_primitive;          /* always MX_OPTMGMT_REQ */
        mx_ulong mx_opt_length;         /* length of options */
        mx_ulong mx_opt_offset;         /* offset of options */
        mx_ulong mx_mgmt_flags;         /* option flags */
} MX_optmgmt_ack_t;

/*
   management flags for MX_OPTMGMT
 */
#define MX_SET_OPT      0x01
#define MX_GET_OPT      0x02
#define MX_NEGOTIATE    0x03
#define MX_DEFAULT      0x04

/*
 *  MX_ATTACH_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_attach_req {
        mx_ulong mx_primitive;          /* always MX_ATTACH_REQ */
        mx_ulong mx_addr_length;        /* length of multiplex address */
        mx_ulong mx_addr_offset;        /* offset of multiplex address */
        mx_ulong mx_flags;              /* options flags */
} MX_attach_req_t;

/*
 *  MX_DETACH_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_detach_req {
        mx_ulong mx_primitive;          /* always MX_DETACH_REQ */
} MX_detach_req_t;

/*
 *  MX_OK_ACK
 *  -------------------------------------------------------------------------
 */
typedef struct MX_ok_ack {
        mx_ulong mx_primitive;          /* always MX_OK_ACK */
        mx_ulong mx_correct_prim;       /* correct primitive */
        mx_ulong mx_state;              /* resulting state */
} MX_ok_ack_t;

/*
 *  MX_ERROR_ACK
 *  -------------------------------------------------------------------------
 */
typedef struct MX_error_ack {
        mx_ulong mx_primitive;          /* always MX_ERROR_ACK */
        mx_ulong mx_error_primitive;    /* primitive in error */
        mx_ulong mx_error_type;         /* MXI error */
        mx_ulong mx_unix_error;         /* UNIX error */
        mx_ulong mx_state;              /* resulting state */
} MX_error_ack_t;

/*
   error types
 */
#define MXSYSERR        0       /* UNIX system error */
#define MXBADADDR       1       /* Bad address format or content */
#define MXOUTSTATE      2       /* Interface out of state */
#define MXBADOPT        3       /* Bad options format or content */
#define MXBADPARM       4       /* Bad parameter format or content */
#define MXBADPARMTYPE   5       /* Bad paramater structure type */
#define MXBADFLAG       6       /* Bad flag */
#define MXBADPRIM       7       /* Bad primitive */
#define MXNOTSUPP       8       /* Primitive not supported */
#define MXBADSLOT       9       /* Bad multplex slot */

/*
 *  MX_ENABLE_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_enable_req {
        mx_ulong mx_primitive;          /* always MX_ENABLE_REQ */
} MX_enable_req_t;

/*
 *  MX_ENABLE_CON
 *  -------------------------------------------------------------------------
 */
typedef struct MX_enable_con {
        mx_ulong mx_primitive;          /* always MX_ENABLE_CON */
} MX_enable_con_t;

/*
 *  MX_DISABLE_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_disable_req {
        mx_ulong mx_primitive;          /* always MX_DISABLE_REQ */
} MX_disable_req_t;

/*
 *  MX_DISABLE_IND
 *  -------------------------------------------------------------------------
 */
typedef struct MX_disable_ind {
        mx_ulong mx_primitive;          /* always MX_DISABLE_IND */
        mx_ulong mx_cause;              /* cause for disable */
} MX_disable_ind_t;

/*
 *  MX_DISABLE_CON
 *  -------------------------------------------------------------------------
 */
typedef struct MX_disable_con {
        mx_ulong mx_primitive;          /* always MX_DISABLE_CON */
} MX_disable_con_t;

/*
 *  MX_DATA_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_data_req {
        mx_ulong mx_primitive;          /* always MX_DATA_REQ */
        mx_ulong mx_slot;               /* slot within multiplex */
} MX_data_req_t;

/*
 *  MX_DATA_IND
 *  -------------------------------------------------------------------------
 */
typedef struct MX_data_ind {
        mx_ulong mx_primitive;          /* always MX_DATA_IND */
        mx_ulong mx_slot;               /* slot within multiplex */
} MX_data_ind_t;
```

```
/*
 *  MX_CONNECT_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_connect_req {
        mx_ulong mx_primitive;          /* always MX_CONNECT_REQ */
        mx_ulong mx_conn_flags;         /* direction to connect */
        mx_ulong mx_slot;               /* slot within multiplex */
} MX_connect_req_t;

/*
   connect flags
 */
#define MXF_RX_DIR      0x01
#define MXF_TX_DIR      0x02
#define MXF_BOTH_DIR    (MXF_RX_DIR|MXF_TX_DIR)

/*
 *  MX_CONNECT_CON
 *  -------------------------------------------------------------------------
 */
typedef struct MX_connect_con {
        mx_ulong mx_primitive;          /* always MX_CONNECT_CON */
        mx_ulong mx_conn_flags;         /* direction connected */
        mx_ulong mx_slot;               /* slot within multiplex */
} MX_connect_con_t;

/*
 *  MX_DISCONNECT_REQ
 *  -------------------------------------------------------------------------
 */
typedef struct MX_disconnect_req {
        mx_ulong mx_primitive;          /* always MX_DISCONNECT_REQ */
        mx_ulong mx_conn_flags;         /* direction to disconnect */
        mx_ulong mx_slot;               /* slot within multiplex */
} MX_disconnect_req_t;

/*
 *  MX_DISCONNECT_IND
 *  -------------------------------------------------------------------------
 */
typedef struct MX_disconnect_ind {
        mx_ulong mx_primitive;          /* always MX_DISCONNECT_IND */
        mx_ulong mx_conn_flags;         /* direction disconnected */
        mx_ulong mx_cause;              /* cause for disconnection */
        mx_ulong mx_slot;               /* slot within multiplex */
} MX_disconnect_ind_t;

/*
 *  MX_DISCONNECT_CON
 *  -------------------------------------------------------------------------
 */
typedef struct MX_disconnect_con {
        mx_ulong mx_primitive;          /* always MX_DISCONNECT_CON */
        mx_ulong mx_conn_flags;         /* direction disconnected */
        mx_ulong mx_slot;               /* slot within multiplex */
} MX_disconnect_con_t;

/*
 *  MX_EVENT_IND
 *  -------------------------------------------------------------------------
 */
typedef struct MX_event_ind {
        mx_ulong mx_primitive;          /* always MX_EVENT_IND */
        mx_ulong mx_event;              /* event */
        mx_ulong mx_slot;               /* slot within multiplex for event */
} MX_event_ind_t;

#define MX_EVT_DCD_ASSERT        0
#define MX_EVT_DCD_DEASSERT      1
#define MX_EVT_DSR_ASSERT        2
#define MX_EVT_DSR_DEASSERT      3
#define MX_EVT_DTR_ASSERT        4
#define MX_EVT_DTR_DEASSERT      5
#define MX_EVT_RTS_ASSERT        6
#define MX_EVT_RTS_DEASSERT      7
#define MX_EVT_CTS_ASSERT        8
#define MX_EVT_CTS_DEASSERT      9
#define MX_EVT_RI_ASSERT        10
#define MX_EVT_RI_DEASSERT      11
#define MX_EVT_YEL_ALARM        12
#define MX_EVT_BLU_ALARM        13
#define MX_EVT_RED_ALARM        14
#define MX_EVT_NO_ALARM         15

#define MXF_EVT_DCD_ASSERT      (1 <<  0)
#define MXF_EVT_DCD_DEASSERT    (1 <<  1)
#define MXF_EVT_DSR_ASSERT      (1 <<  2)
#define MXF_EVT_DSR_DEASSERT    (1 <<  3)
#define MXF_EVT_DTR_ASSERT      (1 <<  4)
#define MXF_EVT_DTR_DEASSERT    (1 <<  5)
#define MXF_EVT_RTS_ASSERT      (1 <<  6)
#define MXF_EVT_RTS_DEASSERT    (1 <<  7)
#define MXF_EVT_CTS_ASSERT      (1 <<  8)
#define MXF_EVT_CTS_DEASSERT    (1 <<  9)
#define MXF_EVT_RI_ASSERT       (1 << 10)
#define MXF_EVT_RI_DEASSERT     (1 << 11)
#define MXF_EVT_YEL_ALARM       (1 << 12)
#define MXF_EVT_BLU_ALARM       (1 << 13)
#define MXF_EVT_RED_ALARM       (1 << 14)
#define MXF_EVT_NO_ALARM        (1 << 15)

#define MXF_EVT_DCD_CHANGE      (MXF_EVT_DCD_ASSERT|MXF_EVT_DCD_DEASSERT)
#define MXF_EVT_DSR_CHANGE      (MXF_EVT_DSR_ASSERT|MXF_EVT_DSR_DEASSERT)
#define MXF_EVT_DTR_CHANGE      (MXF_EVT_DTR_ASSERT|MXF_EVT_DTR_DEASSERT)
#define MXF_EVT_RTS_CHANGE      (MXF_EVT_RTS_ASSERT|MXF_EVT_RTS_DEASSERT)
#define MXF_EVT_CTS_CHANGE      (MXF_EVT_CTS_ASSERT|MXF_EVT_CTS_DEASSERT)
#define MXF_EVT_RI_CHANGE       (MXF_EVT_RI_ASSERT|MXF_EVT_RI_DEASSERT)

#endif                          /* __SS7_MXI_H__ */
```

A.2 MXI Input-Output Controls Header File Listing

```
#ifndef __SS7_MXI_IOCTL_H__
#define __SS7_MXI_IOCTL_H__

#include <linux/ioctl.h>

#define MX_IOC_MAGIC    'c'

/*
 *  CONFIGURATION
 */
typedef struct mx_config {
        mx_ulong type;                  /* unused */
        mx_ulong encoding;              /* encoding */
        mx_ulong block_size;            /* data block size (bits) */
        mx_ulong samples;               /* samples per block */
        mx_ulong sample_size;           /* sample size (bits) */
        mx_ulong rate;                  /* clock rate (samples/second) */
        mx_ulong tx_channels;           /* number of tx channels */
        mx_ulong rx_channels;           /* number of rx channels */
        mx_ulong opt_flags;             /* options flags */
} mx_config_t;

#if 0
typedef struct mx_ifconfig {
        mx_ulong ifaddr;                /* ppa (card,span,channel) */
        volatile mx_ulong ifflags;      /* interface flags */
#define MX_IF_UP            0x01
#define MX_IF_RX_RUNNING    0x02
#define MX_IF_TX_RUNNING    0x04

        mx_ulong iftype;                /* interface type */
#define MX_TYPE_NONE        0
#define MX_TYPE_V35         1
#define MX_TYPE_DS0         2
#define MX_TYPE_DS0A        3
#define MX_TYPE_E1          4
#define MX_TYPE_T1          5
#define MX_TYPE_ATM         6
#define MX_TYPE_PACKET      7

        mx_ulong ifrate;                /* interface rate */
        mx_ulong ifgtype;               /* interface group (span) type */
#define MX_GTYPE_NONE       0
#define MX_GTYPE_T1         1
#define MX_GTYPE_E1         2
#define MX_GTYPE_J1         3
#define MX_GTYPE_ATM        4
#define MX_GTYPE_ETH        5
#define MX_GTYPE_IP         6
#define MX_GTYPE_UDP        7
#define MX_GTYPE_TCP        8
#define MX_GTYPE_RTP        9
#define MX_GTYPE_SCTP       10
```

```
        mx_ulong ifgrate;               /* interface group (span) rate */
        mx_ulong ifmode;                /* interface mode */
#define MX_MODE_NONE        0
#define MX_MODE_DSU         1
#define MX_MODE_CSU         2
#define MX_MODE_DTE         3
#define MX_MODE_DCE         4
#define MX_MODE_CLIENT      5
#define MX_MODE_SERVER      6
#define MX_MODE_PEER        7
#define MX_MODE_REM_LB      8
#define MX_MODE_LOC_LB      9
#define MX_MODE_LB_ECHO     10
#define MX_MODE_TEST        11

        mx_ulong ifgmode;               /* interface group (span) mode */
#define MX_GMODE_NONE       0
#define MX_GMODE_LOC_LB     1
#define MX_GMODE_REM_LB     2

        mx_ulong ifgcrc;                /* interface group crc */
#define MX_GCRC_NONE        0
#define MX_GCRC_CRC4        1
#define MX_GCRC_CRC5        2
#define MX_GCRC_CRC6        3

        mx_ulong ifclock;               /* interface clock */
#define MX_CLOCK_NONE       0
#define MX_CLOCK_INT        1
#define MX_CLOCK_EXT        2
#define MX_CLOCK_LOOP       3
#define MX_CLOCK_MASTER     4
#define MX_CLOCK_SLAVE      5
#define MX_CLOCK_DPLL       6
#define MX_CLOCK_ABR        7
#define MX_CLOCK_SHAPER     8
#define MX_CLOCK_TICK       9

        mx_ulong ifcoding;
#define MX_CODING_NONE      0
#define MX_CODING_NRZ       1
#define MX_CODING_NRZI      2
#define MX_CODING_AMI       3
#define MX_CODING_B6ZS      4
#define MX_CODING_B8ZS      5
#define MX_CODING_ESF       6
#define MX_CODING_AAL1      7
#define MX_CODING_AAL2      8
#define MX_CODING_AAL5      9
#define MX_CODING_HDB3      10

        mx_ulong ifframing;
#define MX_FRAMING_NONE     0
#define MX_FRAMING_CCS      1
#define MX_FRAMING_CAS      2
```

```
#define MX_FRAMING_SF       3
#define MX_FRAMING_D4       MX_FRAMING_SF
#define MX_FRAMING_ESF      4

        mx_ulong ifblksize;
        volatile mx_ulong ifleads;
#define MX_LEAD_DTR         0x01
#define MX_LEAD_RTS         0x02
#define MX_LEAD_DCD         0x04
#define MX_LEAD_CTS         0x08
#define MX_LEAD_DSR         0x10

        mx_ulong ifbpv;
        mx_ulong ifalarms;
#define MX_ALARM_RED        0x01
#define MX_ALARM_BLU        0x02
#define MX_ALARM_YEL        0x04
#define MX_ALARM_REC        0x08

        mx_ulong ifrxlevel;
        mx_ulong iftxlevel;
#define MX_LEVEL_NONE       0
#define MX_LEVEL_75OHM      1
#define MX_LEVEL_100OHM     2
#define MX_LEVEL_120OHM     3
#define MX_LEVEL_LBO_1      4
#define MX_LEVEL_LBO_2      5
#define MX_LEVEL_LBO_3      6
#define MX_LEVEL_LBO_4      7
#define MX_LEVEL_LBO_5      8
#define MX_LEVEL_LBO_6      9

        mx_ulong ifsync;
#define MX_SYNCS            4
        mx_ulong ifsyncsrc[MX_SYNCS];
} mx_ifconfig_t;
#endif

#define MX_IOCGCONFIG   _IOR(   MX_IOC_MAGIC,   2,  mx_config_t     )
#define MX_IOCSCONFIG   _IOWR(  MX_IOC_MAGIC,   3,  mx_config_t     )
#define MX_IOCTCONFIG   _IOWR(  MX_IOC_MAGIC,   4,  mx_config_t     )
#define MX_IOCCCONFIG   _IOR(   MX_IOC_MAGIC,   5,  mx_config_t     )

/*
 *  STATE
 */

typedef struct mx_statem {
        mx_ulong state;
        mx_ulong flags;
} mx_statem_t;

#define MX_IOCGSTATEM   _IOR(   MX_IOC_MAGIC,   6,  mx_statem_t     )
#define MX_IOCCMRESET   _IOR(   MX_IOC_MAGIC,   7,  mx_statem_t     )

/*
```

```
 *  STATISTICS
 */

typedef struct mx_stats {
        mx_ulong header;
        mx_ulong rx_octets;
        mx_ulong tx_octets;
        mx_ulong rx_overruns;
        mx_ulong tx_underruns;
        mx_ulong rx_buffer_overflows;
        mx_ulong tx_buffer_overflows;
        mx_ulong lead_cts_lost;
        mx_ulong lead_dcd_lost;
        mx_ulong carrier_lost;
} mx_stats_t;

#define MX_IOCGSTATSP   _IOR(   MX_IOC_MAGIC,    8, mx_stats_t      )
#define MX_IOCSSTATSP   _IOWR(  MX_IOC_MAGIC,    9, mx_stats_t      )
#define MX_IOCGSTATS    _IOR(   MX_IOC_MAGIC,   10, mx_stats_t      )
#define MX_IOCCSTATS    _IOW(   MX_IOC_MAGIC,   11, mx_stats_t      )

/*
 *  EVENTS
 */

typedef struct mx_notify {
        mx_ulong events;
} mx_notify_t;

#define MX_IOCGNOTIFY   _IOR(   MX_IOC_MAGIC,   12, mx_notify_t     )
#define MX_IOCSNOTIFY   _IOW(   MX_IOC_MAGIC,   13, mx_notify_t     )
#define MX_IOCCNOTIFY   _IOW(   MX_IOC_MAGIC,   14, mx_notify_t     )

typedef struct mx_mgmt {
        mx_ulong cmd;
} mx_mgmt_t;

#define MX_MGMT_RESET           1

#define MX_IOCCMGMT     _IOW(   MX_IOC_MAGIC,   15, mx_mgmt_t       )

#define MX_IOC_FIRST     0
#define MX_IOC_LAST     15
#define MX_IOC_PRIVATE  32

#endif                          /* __SS7_MXI_IOCTL_H__ */
```

Appendix B MXI Drivers and Modules

There are a number of standard drivers and modules provided by the *OpenSS7 Project* the provide capabilities uilizing the Multiplex Interface.

B.1 MXI Drivers

Drivers that provide the MXI interace fall into two categories:

B.1.1 MXI Pseudo-device Drivers

Pseudo-device drivers that accept or provide the MXI interface for the purpose of providing or controlling access the multiplexed facilities available on a system.

B.1.1.1 Multiplexing Driver—`mx`

The `mx` driver is a pseudo-device multiplexing driver that provides simple multiplexing services between MXI Streams at the lower service interface to MXI Streams at the upper service interface. This multiplexing driver is a simplified form of the `matrix` or `mxmux` drivers.

B.1.1.2 Multiplexing Driver—`mxmux`

The `mxmux` driver is a pseudo-device multiplexing driver that provides simple multiplexing services between MXI Streams at the upper service interface and MXI Streams at the lower service interface. It performs interconnection of MXS user Streams to spans, but does not perform switching between lower service interfaces. This multiplexing driver is a simplified form of the `matrix` driver and super-sets the functionality of the `mx` driver.

B.1.1.3 Switching Matrix Multiplexing Driver—`matrix`

The `matrix` driver is a pseudo-device multiplexing driver that provides complete switching matrix and multiplexing services between CHI or MXI Streams at the upper service interface and CHI or MXI Streams at the lower service interface. It performs forward and inverse multiplexing of channels to spans, and performs pseudo-digital cross-connect and dynamic switching of single-, multi- and full-rate channels within the switching matrix. This driver super-sets the functionality of the `chmux` and `mxmux` drivers.

B.1.2 MXI Device Drivers

Real device drivers that provide the MXI interface for the purpose of accessing multiplexed channels available on a hardware device (e.g. a T1 interface card driver). The MXI interface provides a full abstraction of the underlying device driver. The MXI interface is one of the best ways of developing a device driver in support of a multiplexed medium where discrete channels multiplexed into the medium share common timing and syncrhonization. The hardware example is T1, J1 or E1 spans (or even channelized DS3, E3, or SDH VTs). The software example is RTP, PWE2E, G

B.1.2.1 Device Driver—`v401p`

The `v401p(4)` driver is a real device driver that provides access to 4 T1, J1 or E1 interfaces. It is used primarily by the *OpenSS7 Project* as a G.703/G.704 interface for SS7, BSC, SDLC, HDLC, X.21, or voice.

B.2 MXI Modules

STREAMS pushable modules are an excellent way of adapting a MXS user Stream that conforms to the general concept of a communications multiplex into a complex communications protocol. They are also excellent for providing media conversion. For example, it is possible to push a conversion module onto a MXS user Stream correspondin to a mu-law compressed voice channel and convert the media stream to an A-law compressed voice channel.

B.2.1 Modules that convert MXI

The modules (described in the subsections that follow) convert between a MXI interface at the lower service boundary and a MXI interface at the upper service boundary. Conversion is performed on the media stream rather than between service interfaces.

B.2.1.1 Compression Conversion—`mx-conv`

The `mx-conv` module converts one MXI interface to another MXI interface, performing conversion on the media stream in the process. The module is capable of converting between 14-bit signed or unsigned linear, G.711 A-law compressed PCM and G.711 mu-law compressed PCM.

B.2.2 Modules that convert from MXI

The modules (described in the subsections that follow) convert between a MXI interface at the lower service boundary and another interface at the upper service boundary. Conversion is performed between the service interfaces and might or might not include conversion of the bit stream.

B.2.3 Modules that convert to MXI

The modules (described in the subsections that follow) conver between another interface at the lower service boundary and the MXI interface at the upper service boundary. Conversion is performed between the service interfaces and might or might not include conversion of the bit stream.

B.2.3.1 Real-Time Protocol Module—`rtp`

Appendix C MXI Applications

The multiplex interface is a rather important lowest layer component of a number of *OpenSS7 Project* protocol stacks.

C.1 MXI in Switch Matrix

As illustrated in Figure C.1, the MXI interface provides support for access to the *OpenSS7* soft switching matrix.[1]

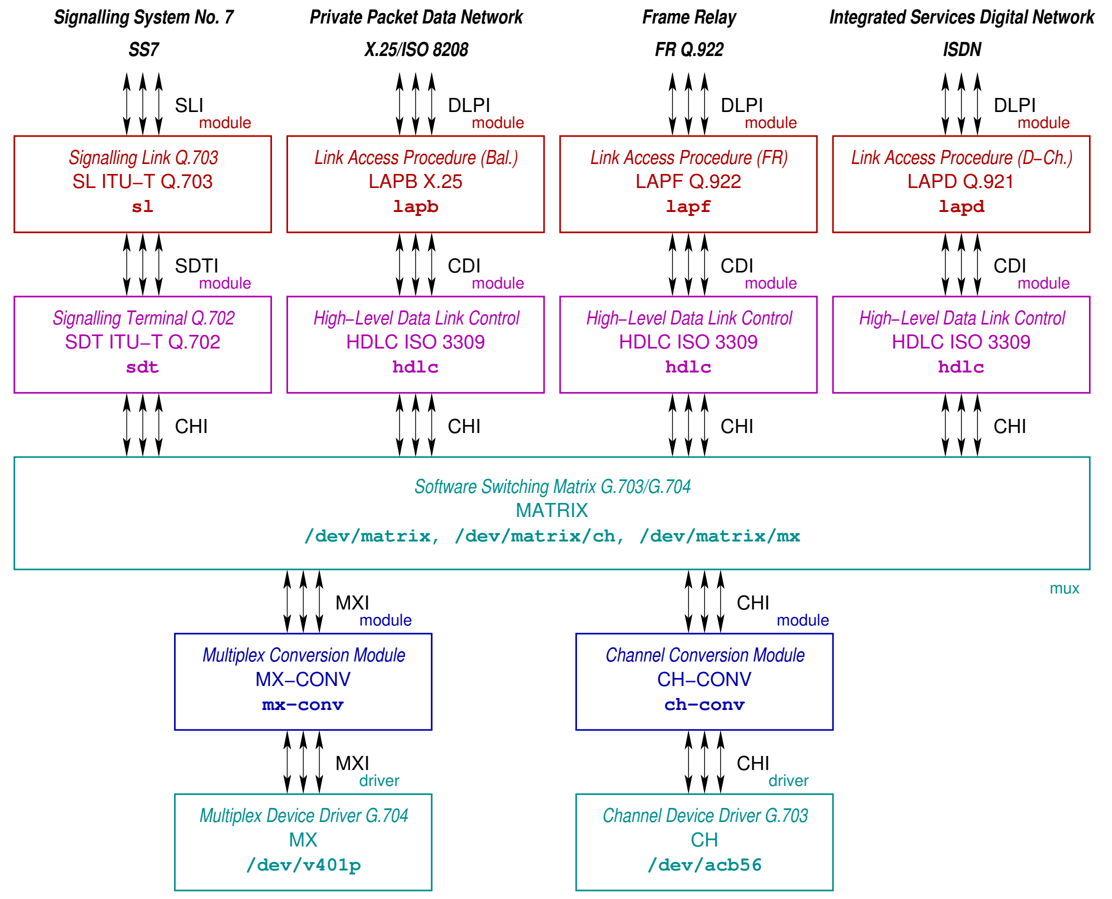
Signalling System No. 7
SS7
Private Packet Data Network
X.25/ISO 8208
Frame Relay
FR Q.922
Integrated Services Digital Network
ISDN
SLI
module
DLPI
module
DLPI
module
DLPI
module
Signalling Link Q.703
SL ITU−T Q.703
sl
Link Access Procedure (Bal.)
LAPB X.25
lapb
Link Access Procedure (FR)
LAPF Q.922
lapf
Link Access Procedure (D−Ch.)
LAPD Q.921
lapd
SDTI
module
CDI
module
CDI
module
CDI
module
Signalling Terminal Q.702
SDT ITU−T Q.702
sdt
High−Level Data Link Control
HDLC ISO 3309
hdlc
High−Level Data Link Control
HDLC ISO 3309
hdlc
High−Level Data Link Control
HDLC ISO 3309
hdlc
CHI
CHI
CHI
CHI
Software Switching Matrix G.703/G.704
MATRIX
/dev/matrix, /dev/matrix/ch, /dev/matrix/mx
mux
MXI
module
CHI
module
Multiplex Conversion Module
MX−CONV
mx-conv
Channel Conversion Module
CH−CONV
ch-conv
MXI
driver
CHI
driver
Multiplex Device Driver G.704
MX
/dev/v401p
Channel Device Driver G.703
CH
/dev/acb56

Figure C.1: *Switch Matrix*

The MXI interface is responsible for providing access to communications channels (single-rate, multi-rate and full-rate) necessary for implementing the synchronous communications channels necessaary for implementing data communications links. Use of the *OpenSS7* software switch matrix at the

[1] A interesting observation is that in Figure C.1, any of the channels that are used for SS7 signalling links, X.25 or OSI links, Frame Relay links or ISDN D-Channel links, can themselves be ISDN B-Channels, E-Channels, H-Channels, or ISUP single-rate or multi-rate IMTs, or even Frame Relay PVCs.

lowest level, as illustrated in Figure C.1, provides a mechanism whereby any synchronous communications channel available to the host can be used as a data communications link, or directly as a voice (or other media) channel.

The switching matrix supports syncrhonous channels using the MXI interface that are one of: single-rate channels, multi-rate channels (statistically multiplexed fractional spans), or full-rate channels (statistically multiplexed full spans). It provides a central point for management of facilities and switching within an *OpenSS7* host and provides for SNMP configuration, monitoring, operational measurements, alarms, events, maintenance access, and other OAM&P functions.

Note also that the MXI interface has the capability of passing synchronous modem lead information to applications as well as Circuit Associated Signalling (A and B bit) and group carrier alarms (Blue, Yellow, Red) for those applications that require them.[2]

C.2 MXI in Zaptel Driver

C.3 MXI in Y.1453 TDM-IP Module

This is a ITU-T Recommendation Y.1453 TDM-IP module. It pushes over a UDP Stream that provides connectivity to the peer TDM-IP system. The upper boundary service interface is the MXI interface. The lower boundary service interface is the UDP-TPI interface.

In general, the UDP Stream may be opened, options configured, bound to a local IP address and port number, and connected to a remote IP address and port number. This module can then be pushed. Pushing the module will flush the Stream and any data messages received on the Stream will be discarded until the Stream is configured, enabled and connected.

Once the module is pushed, the MXI Stream can be linked beneath the MATRIX multiplexing driver and the channels available and the multiplex facility will be made available to the switching matrix.

C.4 MXI in IAX Module

This is an IAX module. It pushes over a UDP Stream that provides connectivity to the peer IAX system. The upper boundary service interface is the MXI interface. The lower boundary service interface is the UDP-TPI interface.

In general, the UDP Stream may be opened, options configured, bound to a local IP address and port number, and connected to a remote IP address and port number. This module can then be pushed. Pushing the module will flush the Stream and any data messages received on the Stream will be discarded until the Stream is configured, enabled and connected.

Once the module is pushed, the MXI Stream can be linked beneath the MATRIX multiplexing driver and the channels available and the multiplex facility will be made available to the switching matrix.

C.5 MXI in SS7 Stack

Figure C.2 illustrates the use of the MXI interface specification in the formation of the SS7 (Signalling System No. 7) protocol stack.

The MXI interface is responsible for providing access to communications channels necessary for implementing signalling data link, signalling terminals and signalling links in accordance with Q.702 and Q.703 as well as similar national standards.

[2] Note that detection of local alarm conditions on carrier facilities is normally required for CAS, ISDN and SS7 ISUP applications where intermediate digital multiplex equipment (i.e. DCCS) can cause distrupt the transparent passing of carrier alarm information between endpoints.

Use of the *OpenSS7* softswitch matrix at the lowest level, as illustrated in Figure C.2, provides a mechanism whereby any communications channel available to the host can be used as an SS7 link.

The major difficulties experienced with such an integrated driver were as follows:

- Because the driver is so closely integrated, it is difficult to use the driver for anything other than SS7 signalling.
- The driver becomes too specific to SS7.
- It becomes difficult to use the devices under this driver approach for voice and switching.
- It becomes difficult to share the device with other applications.
- The SDLI interface does not support fractional (E1/T1) spans.

With the advent of the high-performance *Linux Fast-STREAMS* as well as extremely powerful COTS processors, it is easily possible to separate the protocol levels.[3] Thus, the drivers can provide the generic Multiplex Interface (MXI) that provides direct access to multiplexed spans, or the generic Multiplex Interface (MXI) to provide direct access to non-multiplexed discrete channel devices, and these generic driver interfaces can be linked under the switching matrix multiplexing driver so that a single upper MXI user Stream can provide access to any channel, span, or fractional span within the entire host.

[3] As it turns out, *Linux Fast-STREAMS* has such high performance that higher levels of performance can be acheived by splitting functions into narrowly defined modules that can use STREAMS flow control to keep code path scortching hot.

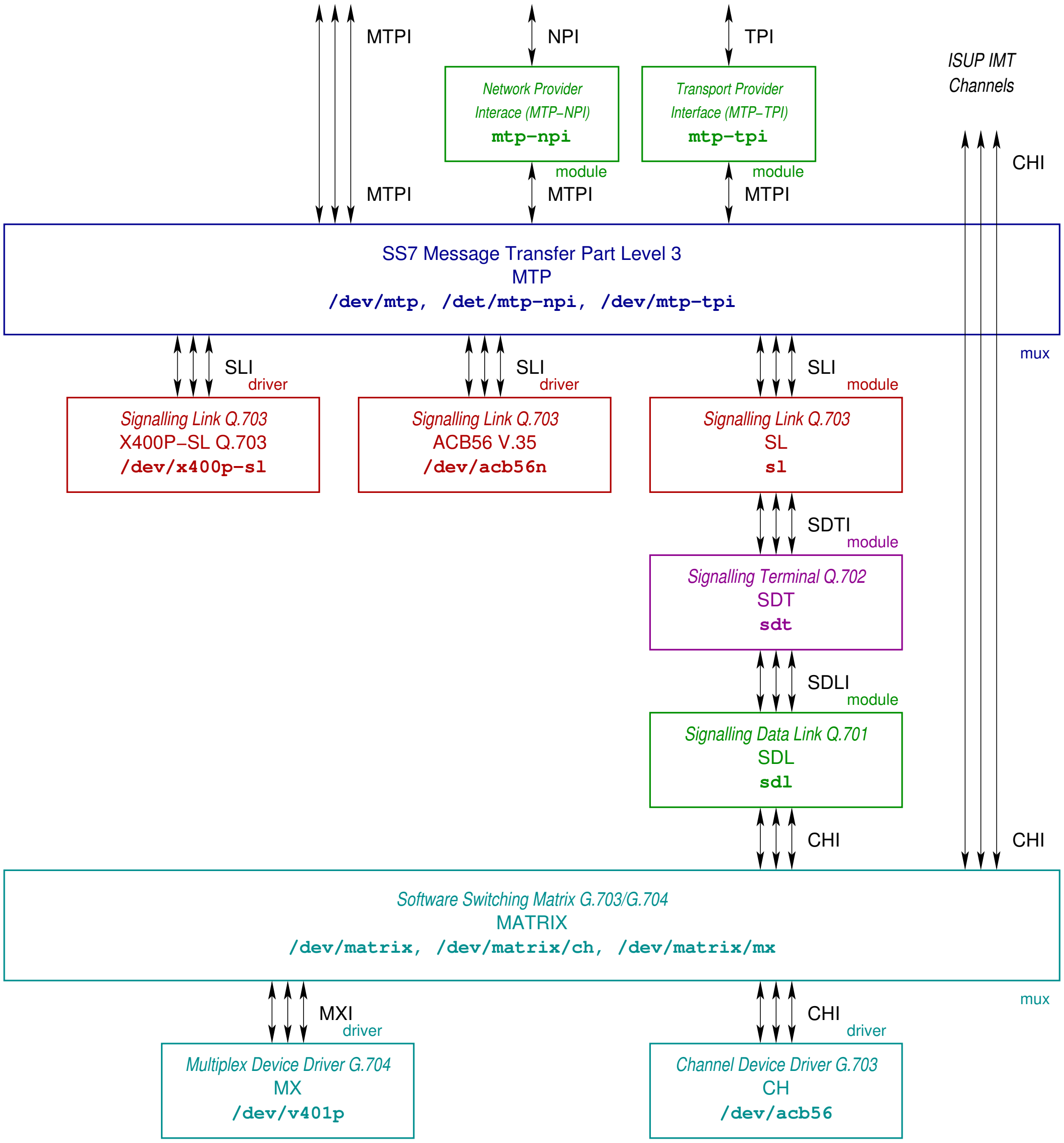

Figure C.2: *SS7 Protocol Stack*

In previous arrangements, the MTP manager opened a Stream on the X400P-SL driver and attached it to a PPA corresponding to either a single-rate channel (Q.703) or a full-rate span (Q.703 Annex B) and linked it beneath the MTP multiplexing driver. This management is not disrupted by the shift to the Software Switching Matrix. A minor device number on the software switching matrix is defined with an autopush specification for the `sdl`, `sdt` and `sl` modules. Opening this minor device number, as before, results in an unattached SL Stream. The MTP manager attaches the Stream as before and links it under the MTP multiplexing driver. This is illustrated in Figure C.2.

C.6 MXI in ISDN Stack

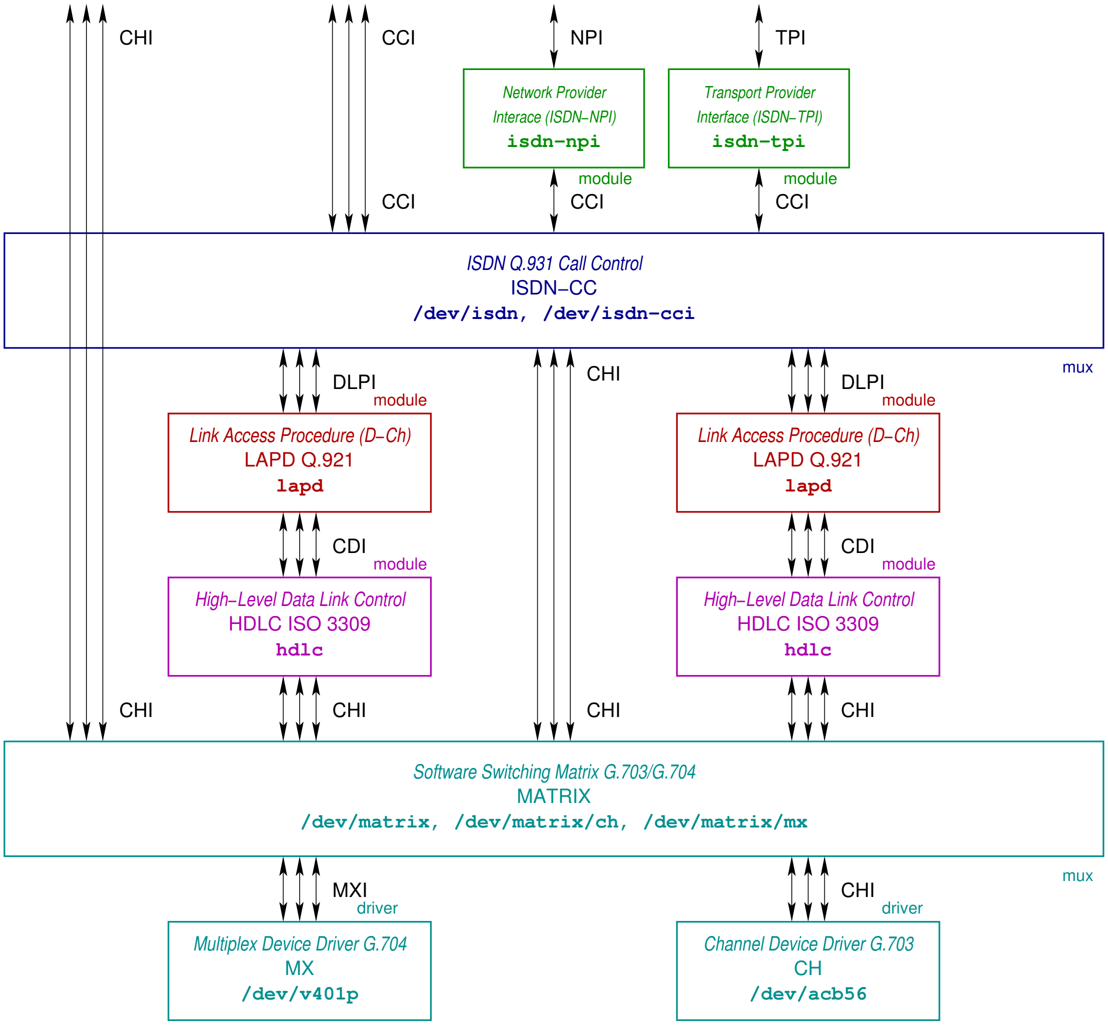

Figure C.3: *ISDN Protocol Stack*

Figure C.3 illustrates the use of the MXI interface specification in the formation of the ISDN (Integrated Services Digital Network) protocol stack. The MXI interface provides two primary categories of access necessary for the ISDN protocol stack:

- Access to multiplexed D channels on the physical medium (either BRI or PRI) for use with HDLC and LAPB protocol modules to form the ISDN signalling link.
- Access to multiplexed B channels on the physical medium (either BPI or PRI) for use with the software switchin matrix `matrix(4)` of media gateway `mg(4)` components. The MXI is also able to provide access to the B-channel provided by CAPI devices.

The MXI interface is responsible for providing switched and permanent access to communications channels necessary for implementing D-channels (HDLC and LAPD) and B-channels (direct access). Use of the *OpenSS7* softswitch matrix at the lowest level, as illustrated in Figure C.3, provides a mechanism whereby any available communications channel available to the host can be used as a D-channel, and any communications channel available to the host can be used as a B-channel.

C.7 MXI in X.25 Stack

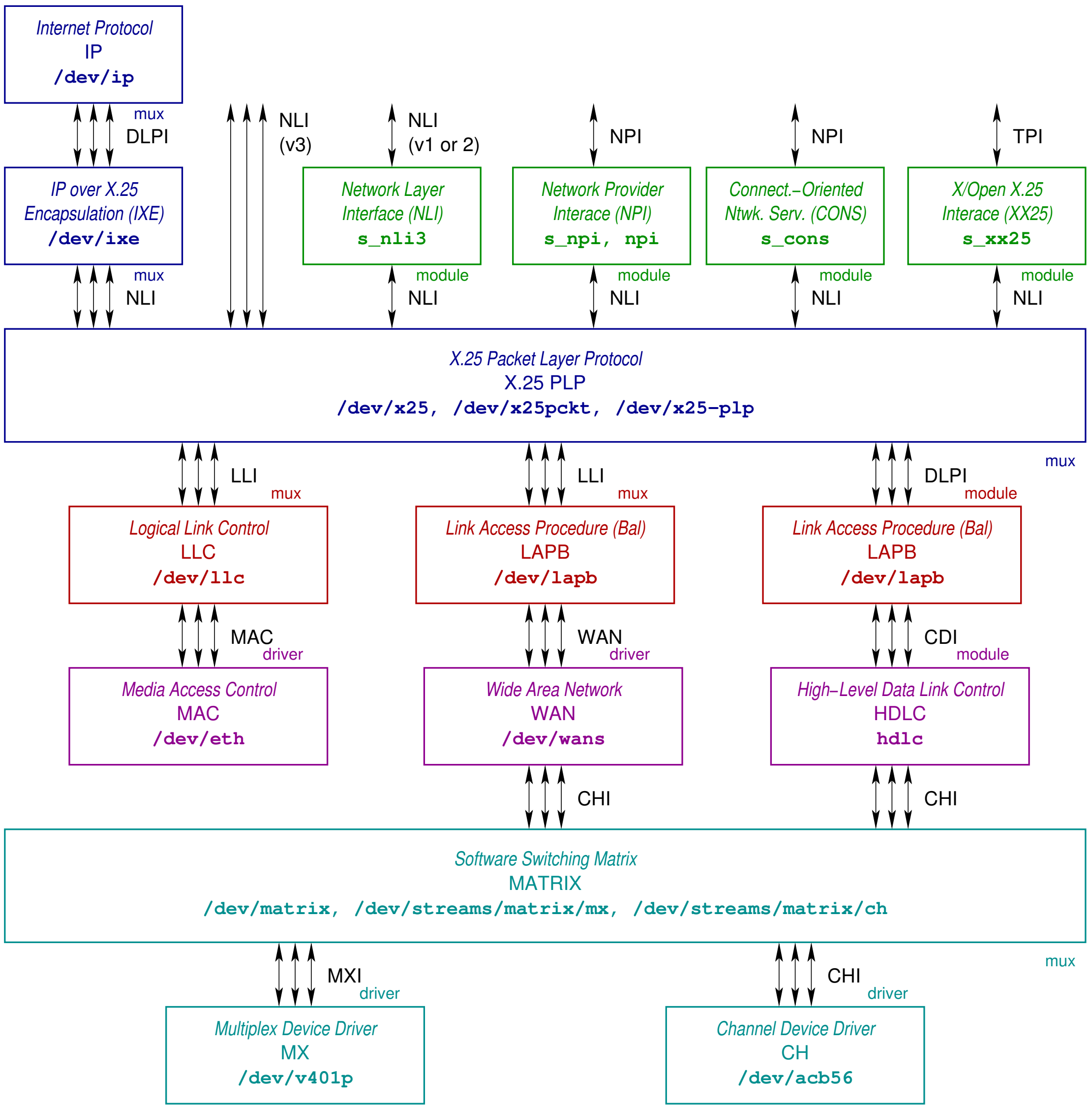

Figure C.4: *X.25 Protocol Stack*

Figure C.4 illustrates the use of the MXI interface specification in the ofrmation of the X.25 protocol stack. The MXI interface provides several primary categories of access necessary for the X.25 protocol stack:

- Access to asyncrhonous modems for dial access to X.25 public or private data networks.
- Access to syncrhonous modems for permanent connections to X.25 public or private data networks.
- Access to ISDN B-channels for switched connections to X.25 public or private data networks.
- Access to channelized, fractional and unchannelized carrier facilities.

The MXI interface is responsible for providing the full and fractional carrier access necessary to perform HDLC and LAPB protocol functions for X.25 and OSI.

Use of the *OpenSS7* softswitch matrix at the lowest level, as illustrated in Figure C.4, provides a mechanism whereby any available communications channel available to the host (including ISDN B-channels) can be used as a LAPB or ISO data link.

C.8 MXI in Frame Relay Stack

As illustrated in Figure C.4, the MXI interface provides support for access to transmission facilities in support of the *OpenSS7* Frame Relay Stack. The MXI interface is responsible for providing the full and fractional carrier access necessary to provide HDLC and LAPF protocol functions for Frame Relay.

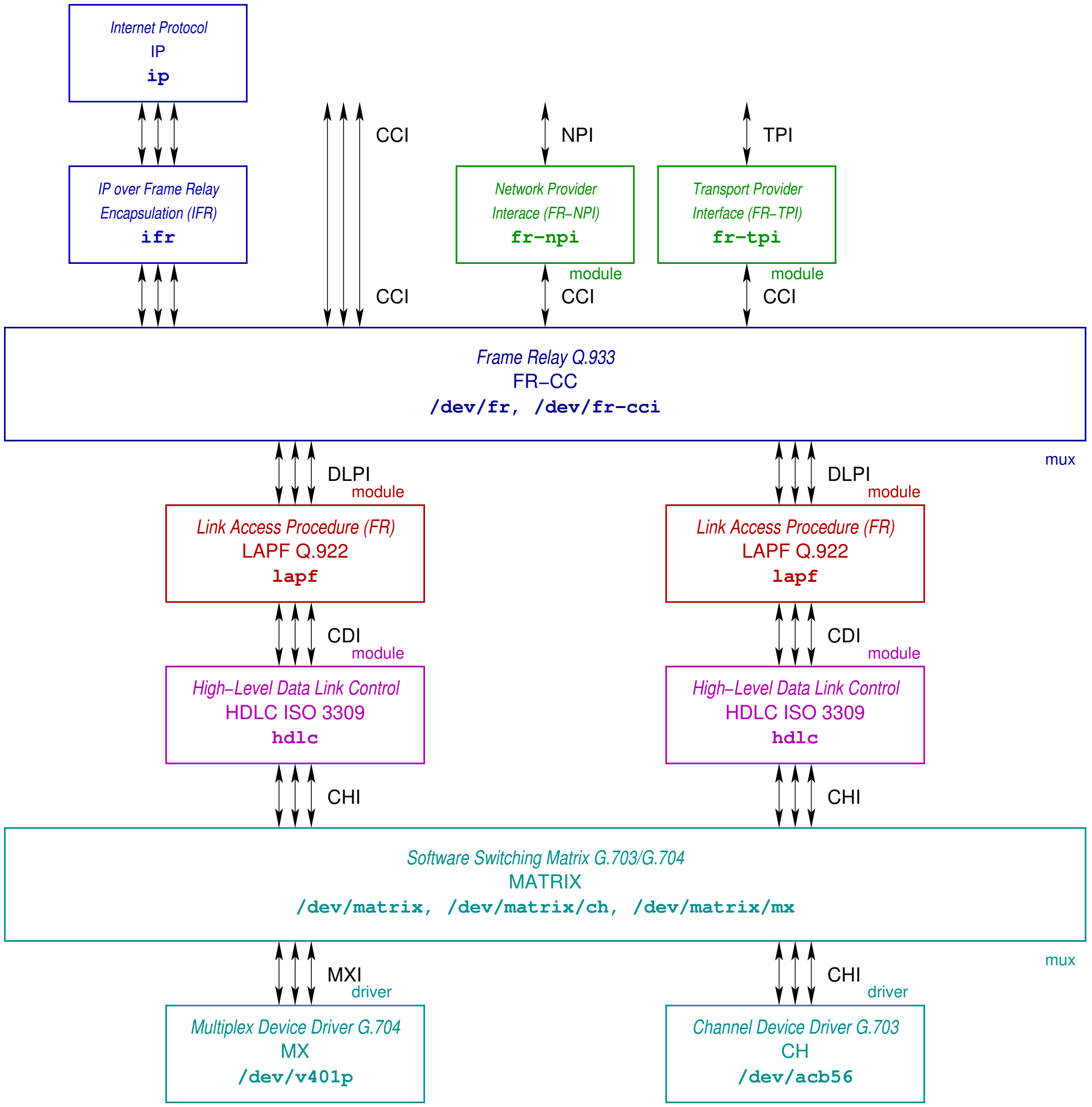

Figure C.5: *Frame Relay Protocol Stack*

Use of the *OpenSS7* softswitch matrix at the lowest level, as illustrated in Figure C.5, provides a mechanism whereby any available communications channel available to the host (including ISDN B-channels) can be used as a Frame Relay data link.

C.9 MXI in Media Gateway

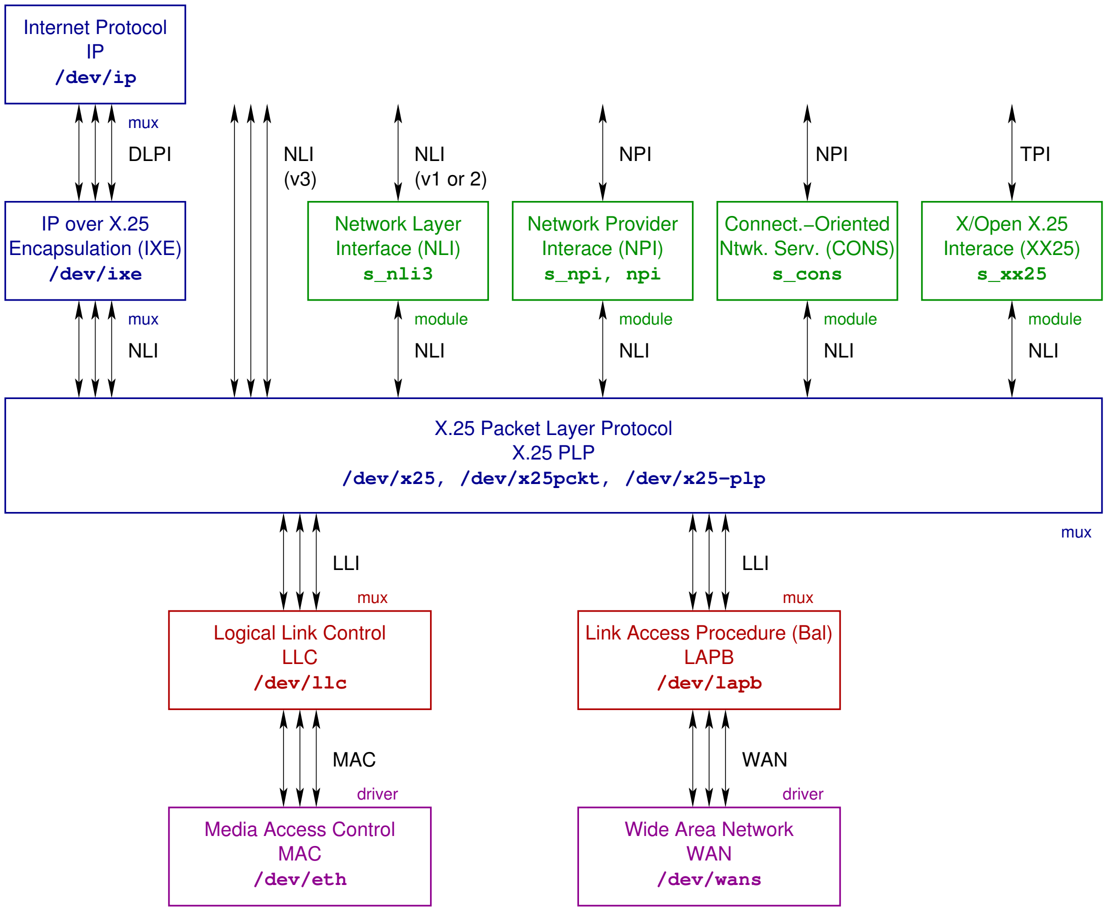

Figure C.6: *Media Gateway*

Appendix D MXI Utilities

Appendix E MXI File Formats

Appendix F MXI Compatibility and Porting

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

AERM	Alignment Error Rate Monitor
CC	Congestion Control
DAEDR	Delimitation Alignment and Error Detection (Receive)
DAEDT	Delimitation Alignment and Error Detection (Transmit)
EIM	Errored Interval Monitor
IAC	Initial Alignment Control
ITU-T	International Telecommunications Union - Telecom Sector
LMS Provider	A provider of Local Management Services
LMS	Local Management Service
LMS User	A user of Local Management Services
LM	Local Management
LSC	Link State Control
PPA	Physical Point of Attachment
RC	Reception Control
SDLI	Signalling Data Link Interface
SDL SDU	Signalling Data Link Service Data Unit
SDLS	Signalling Data Link Service
SDL	Signalling Data Link
SDTI	Signalling Data Terminal Interface
SDTS	Signalling Data Terminal Service
SDT	Signalling Data Terminal
SLI	Signalling Link Interface
SLS	Signalling Link Service
SL	Signalling Link
SL	Signalling Link
SS7	Signalling System No. 7
TXC	Transmission Control

References

[1] ITU-T Recommendation Q.700, *Introduction to CCITT Signalling System No. 7*, March 1993, (Geneva), ITU, ITU-T Telecommunication Standardization Sector of ITU, (Previously "CCITT Recommendation").

[2] ITU-T Recommendation Q.701, *Functional Description of the Message Transfer Part (MTP) of Signalling System No. 7*, March 1993, (Geneva), ITU, ITU-T Telecommunication Standardization Sector of ITU, (Previously "CCITT Recommendation").

[3] ITU-T Recommendation Q.702, *Signalling System No. 7—Signalling Data Link*, March 1993, (Geneva), ITU, ITU-T Telecommunication Standardization Sector of ITU, (Previously "CCITT Recommendation").

[4] ITU-T Recommendation Q.703, *Signalling System No. 7—Signalling Link*, March 1993, (Geneva), ITU, ITU-T Telecommunication Standardization Sector of ITU, (Previously "CCITT Recommendation").

[5] ITU-T Recommendation Q.704, *Message Transfer Part—Signalling Network Functions and Messages*, March 1993, (Geneva), ITU, ITU-T Telecommunication Standardization Sector of ITU, (Previously "CCITT Recommendation").

[6] Geoffrey Gerrietts; Dave Grothe, Mikel Matthews, Dave Healy, *CDI—Application Program Interface Guide*, March 1999, (Savoy, IL), GCOM, Inc.

[7] ITU-T Recommendation Q.771, *Signalling System No. 7—Functional Description of Transaction Capabilities*, March 1993, (Geneva), ITU, ITU-T Telecommunication Standardization Sector of ITU, (Previously "CCITT Recommendation").

Licenses

All code presented in this manual is licensed under the [GNU Affero General Public License], page 111. The text of this manual is licensed under the [GNU Free Documentation License], page 121, with no invariant sections, no front-cover texts and no back-cover texts. Please note, however, that it is just plain wrong to modify statements of, or attribute statements to, the Author or *OpenSS7 Corporation*.

GNU Affero General Public License

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   “This License” refers to version 3 of the GNU Affero General Public License.

   “Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   “The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

   To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

   A “covered work” means either the unmodified Program or a work based on the Program.

   To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

   A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

    Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public 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.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

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
Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If your software can interact with users remotely through a network, you should also make sure that it provides a way for users to get its source. For example, if your program is a web application, its interface could display a "Source" link that leads users to an archive of the code. There are many ways you could offer source, and different solutions will be better for different programs; see section 13 for the specific requirements.

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU AGPL, see http://www.gnu.org/licenses/.

GNU Free Documentation License

GNU FREE DOCUMENTATION LICENSE
Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful 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.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

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. (Thus, 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. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

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 Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document 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, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. 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, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include 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, PostScript or PDF 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.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

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 (or copies in media that commonly have printed covers) 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 computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. 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 fewer than five), unless they release you from this requirement.

   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", Preserve 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. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements 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 to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

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, and that you preserve all their Warranty Disclaimers.

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 “Acknowledgements”, 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, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which 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 half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket 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, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

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. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: 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 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

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.

Index