Message Transfer Part (MTP) Interface (MTPI) Specification

Published by:
OpenSS7 Corporation
1469 Jeffreys Crescent
Edmonton, AB T6L 6T1
Canada

NOTICE

Abstract

This document specifies a Message Transfer Part Interface (MTPI) Specification in support of the OpenSS7 Signalling System No. 7 (SS7) Message Transfer Part (MTP) protocol stacks. It provides abstraction of the MTP-User interface to the services of the Message Transfer Part as well as providing a basis for Integrated Services Digital Network (ISDN) User Part (ISUP), Signalling Connection Control Part (SCCP), Bearer Indexed Call Control (BICC) and other MTP-User protocols.

Preface

Abstract

This document specifies a Message Transfer Part Interface (MTPI) Specification in support of the OpenSS7 Signalling System No. 7 (SS7) Message Transfer Part (MTP) protocol stacks. It provides abstraction of the MTP-User interface to the services of the Message Transfer Part as well as providing a basis for Integrated Services Digital Network (ISDN) User Part (ISUP), Signalling Connection Control Part (SCCP), Bearer Indexed Call Control (BICC) and other MTP-User protocols.

Intent

This document is intended to provide information for writers of OpenSS7 Message Transfer Part Interface (MTPI) applications as well as writers of OpenSS7 Message Transfer Part Interface (MTPI) Users.

This document is intended to provide information for writers of OpenSS7 Message Transfer Part (MTP) applications as well as writers of OpenSS7 Message Transfer Part (MTP) Users. The interface is intended to be used both by upper layer protocol modules as well as by user programs.[1]

Target Audience

The target audience is developers and users of the OpenSS7 SS7 protocol stack.

Disclaimer

Revision History

Take care that you are working with a current version of this document: you will not be informed of updates. For a current version, please see the source documentation at http://www.openss7.org/.

```
$Log: sccpi.me,v $
Revision 0.8.2.1  2003/07/29 00:34:46  brian
Finalizing latest round of drafts.

Revision 0.8  2003/04/30 13:05:28  brian
Started SCCP documentation.

Revision 0.8.2.1  2003/04/22 10:49:54  brian
Working up mtp documentation.

Revision 0.8  2003/02/22 22:37:23  brian
Start of documentation for MTP interface.
```

[1] User programs may find it easier to use the MTP-NPI or MTP-TPI modules which can be pushed over this interface to provide the more familiar Network Provider Interface [NPI] or Transport Provider Interface [TPI].

1. Introduction

This document specifies a STREAMS-based kernel-level instantiation of the ITU-T and ANSI Message Transfer Part primitives [Q.701]. The Message Transfer Part Interface (MTPI) enables the user of MTP services to use any of a variety of conforming MTP service providers without specific knowledge of the provider's protocol variant and variant semantics. The service interface is designed to support any international or national MTP protocol variant. This interface only specifies access to MTP service providers, and does not address issues concerning management, protocol performance, and performance analysis tools. The specification assumes that the reader is familiar with MTP state machines [Q.704] and interfaces [Q.701], and STREAMS.

1.1. Related Documentation [2]

- 1993 ITU-T Q.701 Recommendation [Q.701]
- 1993 ITU-T Q.704 Recommendation [Q.704]
- 2000 ANSI T1.111.1 [T1.111]
- 2000 ANSI T1.111.4 [T1.111]
- System V Interface Definition, Issue 2 − Volume 3 [SVID]

1.1.1. Role

This document specifies an interface that supports the services required by users of the Signalling System No. 7 (SS7) Message Transfer Part (MTP) for International and National applications as described in ITU-T Recommendation Q.700 [Q.700], ETSI Specification XXX, ANSI Specification T1.111 [T1.111], TTC Specification JT.700, and other MTP specifications. These specifications are targeted for use by developers and testers of protocol modules that require Message Transfer Part (MTP) service.

1.2. Definitions, Acronyms, and Abbreviations

1.2.1. Definitions

1.2.2. Acronyms

1.2.3. Abbreviations

[2] See also **References** at the end of this document.

2. The Message Transfer Part

The Message Transfer Part provides the principle means to transfer SS7 messages across an SS7 network. It is responsble for the routing and management of SS7 signalling between MTP-User entities.

2.1. Model of the MTPI

The MTPI defines the services provided by the MTP to the MTP-User at the boundary between the MTP and the MTP-User entity. The interface consists of a set of primitives defined as STREAMS messages that provide access to the MTP services, and are transferred between the MTP-User entity and the MTP. These primitives are of two types; ones that originate from the MTP-User, and others that originate from the MTP. The primitives that originate from the MTP-User make requests to the MTP, or respond to an indication of an event of the MTP. The primitives that originate from the MTP are either confirmations of a request or are indications to the MTP-User that the event has occured. *Figure 2-1* shows the model of the MTPI.

Figure 2-1. Model of the MTPI

The MTPI allows the MTP to be configured with any MTP-User (such as ISUP [Q.764] or SCCP [Q.714]) that also conforms to the MTPI. An MTP-User can also be a user program that conforms to the MTPI and accesses the MTP via **putmsg**(2) and **getmsg**(2) system calls.

2.2. MTPI Services

The features of the MTPI are defined in terms of the services provided by the MTP, and the individual primitives that may flow between the MTP-User and the MTP.

The services supported by the MTPI are based on two distinct modes of communication, connectionless (CLMS) and connection oriented (COMS). Within these modes, the MTPI provides support for both sequenced and unsequenced message transfer. Also, the MTPI supports services for local mangement.

2.2.1. CLMS

The main features of the connectionless mode of communication are:

(1) It is datagram oriented;

(2) It provides transfer of data in self contained units;

(3) There is no logical relationship between these units of data;

Connectionless mode communication has no separate phases. Each unit of data is transmitted from source to destination independently, appropriate addressing information is included with each unit of data. Although the units of data are transmitted independently from source to destination, MTP provides a high level of assurance of sequencing if sequenced service is requested. When unsequenced service is requested, there are no guarantees of proper sequence. Although MTP services are inherently unreliable, MTP provide a high level of assurance that messages are not lost.

The connectionless service of MTP is suited to MTP User protocols such as the Signalling Connection Control Part (SCCP) [Q.711].

2.2.2. COMS

The main features of the MTP connection oriented mode of communication are:

(1) It is virtual circuit oriented.

(2) It provides transfer of data via a pre-established path.

There are three phases to each instance of communication: Connection Establishment, Data Transfer; and Connection Termination. Units of data arrive at their destination in the same order as they departed their source when the sequenced delivery service is requested and the data is protected against duplication or loss of data within some specified quality of service.

The connection oriented service of MTP is suited to MTP User protocols such as the Integrated Services Digital Network User Part (ISUP) [Q.764], Telephone User Part (TUP) [Q.724], and Bearer Indexed Call Control (BICC).[3]

[3]ISUP consists of “signalling relations” between two switches which also have digital facilities between them. In general an ISUP MTP-User can communicate with many other MTP-User peers, however, signalling between any given two enpoints only concerns the digital facilities which exist between the two endpoints. So, management of ISUP switches is best performed on a pairing of endpoints (“signalling relations”). Also, the COMS mode of operation is provided in support of DPC list Routing Keys for M3UA [M3UA].

2.2.3. Local Management

The MTPI specifications also defines a set of local management functions that apply to COMS and CLMS modes of communication. These services have local significance only.

2.2.3.1. Provider Management

The MTPI specification also defines a set of provider management functions that apply to the MTP service provider. These services have local and end-to-end significance.

2.3. MTPI Service Primitives

Table 2-1, *2*, *3* and *4* summarize the MTPI service primitives by their state and service.

Table 2-1. MTPI Service Primitives for Local Management

STATE	SERVICE	PRIMITIVES
Local Management	Information Reporting	MTP_INFO_REQ, MTP_INFO_ACK, MTP_ERROR_ACK
	Bind	MTP_BIND_REQ, MTP_BIND_ACK, MTP_UNBIND_REQ, MTP_OK_ACK, MTP_ERROR_ACK
	Options Management	MTP_OPTMGMT_REQ, MTP_OK_ACK, MTP_ERROR_ACK

Table 2-2. MTPI Service Primitives for Connectionless Mode Data Transfer

STATE	SERVICE	PRIMITIVES
Data Transfer	Data Transfer	MTP_TRANSFER_REQ, MTP_TRANSFER_IND
	Error Management	MTP_PAUSE_IND, MTP_RESUME_IND, MTP_STATUS_IND

Table 2-3. MTPI Service Primitives for Connection Mode Data Transfer

STATE	SERVICE	PRIMITIVES
Connection Establishment	Connection Establishment	MTP_CONN_REQ, MTP_CONN_CON, MTP_OK_ACK, MTP_ERROR_ACK
Data Transfer	Data Transfer	MTP_TRANSFER_REQ, MTP_TRANSFER_IND
	Error Management	MTP_PAUSE_IND, MTP_RESUME_IND, MTP_STATUS_IND
Connection Release	Connection Release	MTP_DISCON_REQ, MTP_OK_ACK, MTP_ERROR_ACK

Table 2-4. MTPI Service Primitives for MTP Management

STATE	SERVICE	PRIMITIVES
Provider Management	Link Management	MTP_INHIBIT_REQ, MTP_INHIBIT_IND, MTP_INHIBIT_CON, MTP_UNINHIBIT_REQ, MTP_UNINHIBIT_IND, MTP_UNINHIBIT_CON, MTP_ERROR_ACK
	Route Management	MTP_PROHIBIT_REQ, MTP_PROHIBIT_IND, MTP_PROHIBIT_CON, MTP_ALLOW_REQ, MTP_ALLOW_IND, MTP_ALLOW_CON, MTP_ERROR_ACK
	Layer Management	MTP_EVENT_IND, MTP_STATS_IND

3. MTPI Services Definition

This section describes the services of the MTPI primitives. Time-sequence diagrams that illustrate the sequence of primitives are included.[4] The format of the primitives will be defined later in this document.

3.1. Local Management Services Definition

The services defined in this section are outside the scope of international standards. These services apply to COMS and CLMS modes of communication. They are invoked for the initialization/de-initialization of a stream connected to the MTP. They are also used to manage options supported by the MTP and to report information on the supported parameter values.

3.1.1. Message Transfer Part Information Reporting Service

This service provides information on the options supported by the MTP provider.

- **MTP_INFO_REQ**: This primitive requests that the MTP return the values of all the supported protocol parameters. This request may be invoked during any phase.
- **MTP_INFO_ACK**: This primitive is in response to the MTP_INFO_REQ primitive and returns the values of the supported protocol parameters to the MTP-User.

The sequence of primitive for MTP information management is shown in *Figure 3-1*.

Figure 3-1. Sequence of Primitives: Message Transfer Part Information Reporting Service

3.1.2. MTP Address Service

This service allows an MTP-User to determine the MTP address (MTP-SAPI or signalling point code and service indicator) that has been associated with a stream. It permits the MTP-User to not necessarily retain this information locally, and allows the MTP-User to determine this information from the MTP provider at any time.

- **MTP_ADDR_REQ**: This primitive requests that the MTP return information concerning which MTP address (MTP-SAPI) the MTP-User is bound as well as the MTP address upon which the MTP-User is currently engaged in association.
- **MTP_ADDR_ACK**: This primitive is in response to the MTP_ADDR_REQ primitive and indicates to the MTP-User the requested information.

The sequence of primitives is shown in *Figure 3-2*.

Figure 3-2. Sequence of Primitives: Message Transfer Part User Address Service

3.1.3. MTP User Bind Service

This service allows an MTP address (MTP-SAPI: signalling point code and service indicator) to be associated with a stream.

- **MTP_BIND_REQ**: This primitive requests that the MTP-User be bound to a particular MTP address (MTP-SAPI), and negotiate the number of allowable outstanding setup indications for that address.
- **MTP_BIND_ACK**: This primitive is in response to the MTP_BIND_REQ primitive and indicates to the user that the specified MTP-User has been bound to an MTP address.

The sequence of primitives is shown in *Figure 3-3*.

Figure 3-3. Sequence of Primitives: Message Transfer Part User Bind Service

3.1.4. MTP User Unbind Service

This service allows the MTP-User to be unbound from an MTP address.

- **MTP_UNBIND_REQ**: This primitive requests that the MTP-User be unbound from the MTP address that it had previously been bound to.

[4] Conventions for the time-sequence diagrams are defined in ITU-T X.210 [X.210].

The sequence of primitives is shown in *Figure 3-4*.

Figure 3-4. Sequence of Primitives: Message Transfer Part User Unbind Service

3.1.5. Receipt Acknolwedgement Service

- **MTP_OK_ACK**: This primitive indicates to the MTP-User that the previous MTP-User originated primitive was received successfully by the MTP.

An example showing the sequence of primitives for successful receipt acknowledgement is depicted in *Figure 3-5*.

Figure 3-5. Sequence of Primitives: Message Transfer Part Receipt Ackknowledgement Service

3.1.6. Options Management Service

This service allows the MTP-User to manage options parameter values associated wtih the MTP.

- **MTP_OPTMGMT_REQ**: This primitive allows the MTP-User to select default values for options parameters within the range supported by the MTP, and to indicate the default selection of receipt confirmation.

Figure 3-6 shows the sequence of primitives for MTP options management.

Figure 3-6. Sequence of Primitives: Message Transfer Part Options Management Service

3.1.7. Error Acknolwedgement Service

- **MTP_ERROR_ACK**: This primitive indicates to the MTP-User that a non-fatal error has occured in the last MTP-User originated request or response primitive (listed in *Figure 3-7*), on the stream.

Figure 3-7 shows the sequence or primitives for the error management primitive.

Figure 3-7. Sequence of Primitives: Message Transfer Part Error Acknowledgement Service

3.2. Connectionless Services Definition

The CLMS allows for the transfer of MTP-User data in one or both directions simultaneously without establishing an association between MTP-User peers. A set of primitives are defined that carry user data and control information between the MTP-User and MTP entities. The primitives are modeled as requests initiated by the MTP-User and indications initiated by the MTP provider. Indications may be initiated by the MTP independently from requests by the MTP-User.

The connectionless MTP service consists of one phase.

3.2.1. Data Transfer

3.2.1.1. User Primitives for Data Transfer

- **MTP_TRANSFER_REQ**: This primitive requests that the MTP send the data unit to the specified destination with the specified sequence control.

3.2.1.2. Provider Primitives for Data Transfer

- **MTP_TRANSFER_IND**: This primitive indicates to the MTP-User that a data unit has been received from the specified source address.

Figure 3-8 shows the sequence of primitives for the connectionless mode of data transfer.

Figure 3-8. Sequence of Primitives: Message Transfer Part Data Transfer

3.2.2. Error Management

3.2.2.1. Provider Primitives for Error Management

- **MTP_PAUSE_IND**: This primitive indicates to the MTP-User that the specified destination address is no longer accessible.
- **MTP_RESUME_IND**: This primitive indicates to the MTP-User that the specified destination address is now accessible.
- **MTP_STATUS_IND**: This primitive indicates ot the MTP-User that the congestions status to the specified destination address has changed, or that the remote MTP-User is no longer available.

Figure 3-9 shows the sequence of primitives for the connectionless mode error management primitives.

Figure 3-9. Sequence of Primitives: Message Transfer Part Error Management

3.3. Connection Oriented Services Definition

This section describes the required MTP service primitives that define the CLMS interface.

The queue model for CLMS is discused in more detail in ITU-T Q.704 [Q.704]. For Q.704 specific conformance considerations, see Addendum 1.

The queue model represents the operation of an MTP connection in the abstract by a pair of queues linking the two MTP addresses. There is one queue for each direction of signalling transfer. The ability of a user to add objects to a queue will be determined by the behavior of the user removing objects from that queue, and the state of the queue. The pair of queues is considered to be available for each potential association. Objects that are entered or removed from the queue are either as a result of interactions at the two MTP addresses, or as the result of MTP initiatives.

- A queue is empty until a connect object has been entered and can be returned to this state, with loss of its contents, by the MTP.
- Objects may be entered into a queue as a result of the action of the source MTP-User, subject to control by the MTP.
- Objects may also be entered into a queue by the MTP.
- Objects are removed from the queue under the control of the receiving MTP user.
- Objects are normally removed under the control of the MTP-User in the same order as they were entered except:
  - if the object is of a type defined to be able to advance ahead of the preceding object (however, no object is defined to be able to advance ahead of another object of the same type), or
  - if the following object is defined to be destructive with respect to the preceding object on the queue. If necessary, the last object on the queue will be deleted to allow a destructive object to be entered − they will therefore always be added to the queue. For example, "reset" objects are defined to be destructive with respect to all other objects.

Table 3-1 shows the ordering relationship amoung the queue model objects.

Table 3-1. Flow Control Relationships Between Queue Model Objects

Object X Object Y	CONNECT	DATA	MANAGEMENT	DISCONNECT
CONNECT	N/A	−	−	DES
DATA	N/A	−	AA	DES
MANAGEMENT	N/A	−	−	DES
DISCONNECT	N/A	N/A	N/A	−

AA	Indicates that Object X is defined to be able to advance ahead of preceding Object Y.
DES	Indicates that Object X is defined to be destructive with respect to preceding Object Y.
−	Indicates that Object X is neither destructive with respect to Object Y, nor able to advance ahead of Object Y.
N/A	Indicates that Object X will not occur in a position succeeding Object Y in a valid state of a queue.

3.3.1. Connection Establishment Phase

A pair of queues is associated with an MTP association between two MTP addresses when the MTP receives an MTP_CONN_REQ primitive at one of the MTP addresses resulting in a connect object being entered into the queue. The queues will remain associated with the MTP association until a N_DISCON_REQ primitive (resulting in a disconnect object) is either entered or removed from a queue. Similarly, in the queue from the remote MTP-User, objects can be entered into the queue only after the connect object associated with an MTP_CONN_REQ has been entered into the queue.

The MTP association procedure will fail if the MTP is unable to route to the remote MTP-User.

3.3.1.1. User primitives for Successful MTP Association Establishment

- **MTP_CONN_REQ**: This primitive requests that the MTP establish an association between the local MTP-User and the specified destination.

3.3.1.2. Provider primitives for Successful MTP Association Establishment

- **MTP_CONN_CON**: This primitive indicates to the MTP-User that an association request has been confirmed.

The sequence of primitives in a successful MTP association establishment is defined by the time sequence diagram as shown in *Figure 3-10*.

Figure 3-10. Sequence of Primitives: Message Transfer Part Association Service

3.3.2. Data Transfer Phase

Flow control on the MTP association is done by management of queue capacity, by allowing objects of certain type to be inserted to the queues as shown in *Table 3-2*.

3.3.2.1. User primitives for MTP Data Transfer

- **MTP_TRANSFER_REQ**: This primitive requests that the MTP transfer the specified data.

3.3.2.2. Provider primitives for MTP Data Transfer

- **MTP_TRANSFER_IND**: This primitive indicates to the MTP-User that this message contains data.

Figure 3-11 shows the sequence of primitive for successful data transfer. The sequence of primitive may remain complete if an MTP_DISCON_REQ primitive occurs.

Figure 3-11. Sequence of Primitives: Message Transfer Part Data Transfer

This sequence of primtives may remain incomplete if an MTP_PAUSE or MTP_STATUS indication is received from the MTP.

3.3.3. Error Management Primitives

The MTP error management service is used by the MTP to report detected loss of unrecoverable data.

3.3.3.1. Provider Primitives for Management

- **MTP_PAUSE_IND**: This primitive indicates to the MTP-User that the remote MTP is no longer accessible.
- **MTP_RESUME_IND**: This primitive indicates to the MTP-User that the remote MTP is now accesible.
- **MTP_STATUS_IND**: This pirmitive indicates to the MTP-User that the congestion status to te remote MTP has changed, or that the remote MTP User is no longer accessible.

Figure 3-12 shows the sequence of primitives for the connection mode error management primitives. The sequence of primitives may remain complete if an MTP_DISCON primitive occurs.

Figure 3-12. Sequence of Primitives: Message Transfer Part Error Management

3.3.4. Connection Termination Phase

The MTP association release procedure is initialized by the insertion of a disconnect object (associated with an MTP_DISCON_REQ) into the queue. As shown in *Table 3-1*, the disconnect procedure is destructive with respect to other objects in the queue, and eventually results in the emptying of queues and termination of the MTP association.

3.3.4.1. User Primitives for MTP Assocation Termination

- **MTP_DISCON_REQ**: This primitive requests that the MTP disconnect an exising MTP association.

The sequence of primitives are shown in the time sequence diagram in *Figure 3-13*.

Figure 3-13. Sequence of Primitives: Message Transfer Part Connection Termination

3.4. MTP Provider Management Primitives

This section describes the required MTP service primitives that define the MTP Provider Management interface.

MTP Provider Management allows for the coordination of MTP mangement messages between MTP Provider peers. A set of primitives are defined that invoke management actions which are communicated from MTP to MTP entities. The primitives are modeled as requires initiated by the MTP management and indications initiated by the MTP. Indications may be initiated by the MTP independently from requests by the MTP management.

The MTP Provider Management service consists of one phase.

3.4.1. Link Management

The MTP link management service allows MTP management to inhibit or uninhibit a link, linkset or combined linkset.

3.4.1.1. User Primitives for Link Inhibit Service

- **MTP_INHIBIT_REQ**: Requests that the MTP inhibit the specified link, linkset or combined linkset.

3.4.1.2. Provider Primitives for Link Inhibit Service

- **MTP_INHIBIT_IND**: Indicates that the remote MTP has inibited the specified link, linkset or combined linkset.
- **MTP_INHIBIT_CON**: Confirms that the MTP has successfully inhibited the specified link, linkset or combined linkset in coordination with the MTP peer.

Figure 3-14 shows the sequence of primitives for the MTP management link inihibit service, for a successful link inhibit.

Figure 3-14. Sequence of Primitives: Message Transfer Part Successful Link Inhibit

Figure 3-15 shows the sequence of primitives for the MTP management link inihibit service, for an unsuccessful link inhibit.

Figure 3-15. Sequence of Primitives: Message Transfer Part Unsuccessful Link Inhibit

3.4.1.3. User Primitives for Link Uninhibit Service

- **MTP_UNINHIBIT_REQ**: Requests that the MTP uninhibit the specified link, linkset or combined linkset.

3.4.1.4. Provider Primitives for Link Uninhibit Service

- **MTP_UNINHIBIT_IND**: Indicates that an inhibit attempt has failed or the remote MTP has uninibited the specified link, linkset or combined linkset.
- **MTP_UNINHIBIT_CON**: Confirms that the MTP has successfully uninhibited the specified link, linkset or combined linkset in coordination with the MTP peer.

Figure 3-16 shows the sequence of primitives for the MTP management link uninihibit service.

Figure 3-16. Sequence of Primitives: Message Transfer Part Link Uninhibit Service

3.4.2. Route Management

The MTP route management service allows MTP management to allow or prohibit a route or routeset to a specific destination.

3.4.2.1. User Primitives for Route Allow Service

- **MTP_ALLOW_REQ**: Requests that the MTP allow the specified route or routeset for the specified destination.

3.4.2.2. Provider Primitives for Route Allow Service

- **MTP_ALLOW_IND**: Indicates to the MTP-User that the network has allowed the specified route or routeset for the specified destination.
- **MTP_ALLOW_CON**: Confirms to the MTP-User that the MTP has successfully allowed the specified route or routeset for the specified destination to the network.

Figure 3-17 shows the sequence of primitives for the MTP management route allow service.

Figure 3-17. Sequence of Primitives: Message Transfer Part Route Allow Service

3.4.2.3. User Primitives for Route Prohibit Service

- **MTP_PROHIBIT_REQ**: Requests that the MTP prohibit the specified route or routeset for the specified destination.

3.4.2.4. Provider Primitives for Route Prohibit Service

- **MTP_PROHIBIT_IND**: Indicates to the MTP-User that an allow request has failed or the network has prohibitted the specified route or routeset for the specified destination.
- **MTP_PROHIBIT_CON**: Confirms to the MTP-User that the MTP has successfully prohibited the specified route or routeset for the specified destination to the newtork.

Figure 3-18 shows the sequence of primitives for the MTP management route prohibit service.

Figure 3-18. Sequence of Primitives: Message Transfer Part Route Prohibit Service

3.4.3. Layer Management

The MTP layer management service allows MTP management to request and receive event indications (alarms and 1st and deltas) and to request and receive stats indications (statistics and operational measurements) for specified intervals.

3.4.3.1. User Primitives for Event Indications

- **MTP_NOTIFY_REQ**: Requests that the MTP issue notifications for specified events.

3.4.3.2. Provider Primitives for Event Indications

- **MTP_EVENT_IND**: Indicates to the MTP-User a notification of a requested event.

Figure 3-19 shows the sequence of primitives for the MTP management event service.

Figure 3-19. Sequence of Primitives: Message Transfer Part Event Service

3.4.3.3. User Primitives for Statistics Indications

- **MTP_STATS_REQ**: Requests that the MTP collect specified statistics and operational measurements on specified intervals.

3.4.3.4. Provider Primitives for Statistics Indications

- **MTP_STATS_IND**: Indicates to the MTP-User that the specified statistics and operational measurements have been collected as requested.

Figure 3-20 shows the sequence of primitives for the MTP management statistics service.

Figure 3-20. Sequence of Primitives: Message Transfer Part Measurements Service

4. MTPI Primitives

This section describes the format and parameters of the MTPI primitives (Appendix A shows the mapping of MTPI primitives fo the primitives defined in Q.701 [Q.701] and T1.111.1 [T1.111]). Also, it discussses the states the primitive is valid in, the resulting state, and the acknowledgement that the primitive expects. (The state/event tables for these primitives are shown in Appendix B. The precedence tables for the MTPI primitives are shown in Appendix C.) Rules for ITU-T conformance [Q.704] are described in Addendum 1 to this document, rules for ANSI conformance [T1.111] are described in Addendum 2, and rules for JITC conformance [JT-Q.704] are described in Addendum 3.

Table 4-1, *2* and *3* provide a summary of the MTP primitives and their parameters.

4.1. Local Management Primitives

These primitives apply to CLMS and COMS.

4.1.1. Message Transfer Part Information Request

MTP_INFO_REQ

This primitive requests the MTP to return the values of all supported protocol parameters (see under MTP_INFO_ACK), and also the current state of the MTP (as defined in Appendix B). This primitive does not affect the state of the MTP and does not appear in the state tables.

Format

The format of the message is one M_PCPROTO message block and its structure is as follows:

```
typedef struct MTP_info_req {
        mtp_ulong mtp_primitive;        /* always MTP_INFO_REQ */
} MTP_info_req_t;
```

Parameters

mtp_primitive:	Indicates the primitive type. This field is always coded MTP_INFO_REQ.

Valid States

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

New State

The new state remains unchanged.

Acknowledgements

This primitive requires the MTP to generate one of the following acknowledgements upon receipt of the primitive:

- **Successful**: Acknowledgement of the primitive via the MTP_INFO_ACK primitive.
- **Non-fatal errors**: There are no errors associated with the issuance of this primitive.

4.1.2. Message Transfer Part Information Acknowledgement

MTP_INFO_ACK

This primitive indicates to the MTP-User any relevant protocol-dependent parameters. It should be initiated in response to the MTP_INFO_REQ primitive described above.

Format

The format of this message is one M_PCPROTO message block and its structure is as follows:

```
typedef struct MTP_info_ack {
        mtp_ulong mtp_primitive;        /* always MTP_INFO_ACK */
        mtp_ulong mtp_msu_size;         /* maximum MSU size for guaranteed delivery */
        mtp_ulong mtp_addr_size;        /* maximum address size */
        mtp_ulong mtp_addr_length;      /* address length */
        mtp_ulong mtp_addr_offset;      /* address offset */
        mtp_ulong mtp_current_state;    /* current interface state */
        mtp_ulong mtp_serv_type;        /* service type */
        mtp_ulong mtp_version;          /* version of interface */
} MTP_info_ack_t;

#define M_COMS  1               /* Connection-mode MTP service supported */
#define M_CLMS  2               /* Connection-less MTP service supported */
```

Parameters

The above fields have the following meaning:

mtp_primitive:	Indicates the primitive type. This field is always coded MTP_INFO_ACK.
mtp_msu_size:	Indicates the maximum MSU size.
mtp_addr_size:	Indicates the (maximum) size of an MTP address (MTP-SAPI).
mtp_addr_length:	Indicates the length of the bound and connected MTP addresses. When the stream is in a bound state, the MTP addresses include the bound address. When the stream is in a connected state and the service type is M_COMS, the MTP addresses also includes the connected address.
mtp_addr_offset:	Indicates the offset of the bound and connected MTP addresses from the start of the M_PCPROTO block.
mtp_current_state:	Indicates the current state of the MTP interface.
mtp_serv_type:	Indicates the service type of the interface. The service type is either M_COMS or M_CLMS.
mtp_version:	Indicates the version of the interface.

Flags

Valid States

This primitive is valid in any state in response to an MTP_INFO_REQ primitive.

New State

The state remains the same.

4.1.3. Protocol Address Request

MTP_ADDR_REQ

This primitive requests that the MTP return information concerning the MTP addresses upon which the MTP-User is bound or engaged in an association.

The format of the message is one M_PROTO message block and its structure is as follows:

```
typedef struct MTP_addr_req {
        mtp_ulong mtp_primitive;        /* always MTP_ADDR_REQ */
} MTP_addr_req_t;
```

Parameters

mtp_primitive:	Specifies the primitive type. This field is always coded MTP_ADDR_REQ.

Valid States This primitive is valid in any state.

New State The new state is MTPS_WACK_AREQ.

Rules

- If the stream is not in a bound state, no bound or connected address information will be returned in the MTP_ADDR_ACK.
- If the stream is bound and not connected, no connected address information will be returned in the MTP_ADDR_ACK.

Acknowledgements

The MTP will generate on of the following acknowledgements upon receipt of the MTP_ADDR_REQ primitive:

- **Successful**: Correct acknowledgment of the primitive is indicated via the MTP_ADDR_ACK primitive.
- **Unsuccessful (Non-fatal errors)**: These errors will be indicated via the MTP_ERROR_ACK primitive. The applicable non-fatal errors are as follows:

MSYSERR:	A system error occured and the UNIX system error is indicated in the primitive.

4.1.4. Protocol Address Acknowledgement

MTP_ADDR_ACK

This primitive acknowledges the corresponding request primitive and is used by the MTP to return information concerning the local and remote protocol addresses for the stream.

The format of the message is one M_PCPROTO message block and its structure is as follows:

```
typedef struct MTP_addr_ack {
        mtp_ulong mtp_primitive;        /* always MTP_ADDR_ACK */
        mtp_ulong mtp_loc_length;       /* length of local MTP address */
        mtp_ulong mtp_loc_offset;       /* offset of local MTP address */
        mtp_ulong mtp_rem_length;       /* length of remote MTP address */
        mtp_ulong mtp_rem_offset;       /* offset of remote MTP address */
} MTP_addr_ack_t;
```

Parameters

mtp_primitive:	Indicates the primitive type. This field is always coded MTP_ADDR_ACK.
mtp_loc_length:	Indicates the length of the local MTP address (MTP-SAPI). If the stream is not bound to a local MTP-SAPI, this field will be coded zero (0).
mtp_loc_offset:	Indicates the offset of the local MTP address (MTP-SAPI) from the start of the M_PCPROTO message block. If the stream is not bound to a local MTP-SAPI, this field will be coded zero (0).
mtp_rem_length:	Indicates the length of the remote MTP address (MTP-SAPI). If the stream is not connected to a remote MTP-SAPI, this field will be coded zero (0).
mtp_rem_offset:	Indicates the offset of the remote MTP address (MTP-SAPI) from the start of the M_PCPROTO message block. If the stream is not connected to a remote MTP-SAPI, this field will be coded zero (0).

Valid State

This primitive is valid in state MTP_WACK_AREQ.

New State

The new state is the state previous to the MTP_ADDR_REQ.

Rules

- If the requesting stream is not bound to an MTP address (MTP-SAPI), the MTP will code the *mtp_loc_length* and *mtp_loc_offset* fields to zero. Otherwise, the MTP will return the same MTP address that was returned in the MTP_BIND_ACK.
- If the requesting stream is not associated with a remote peer (i.e, not bound in a signalling relation), the MTP will code the *mtp_rem_length* and *mtp_rem_offset* fields to zero. Otherwise, the MTP will indicate the remote MTP address (MTP-SAPI) of the associated remote MTP-User.

4.1.5. Bind Protocol Address Request

MTP_BIND_REQ

This primitive requests that the MTP bind an MTP-User entity to an MTP address (MTP-SAPI).

Format

The format of the message is one M_PROTO message block and its structure is as follows:

```
typedef struct MTP_bind_req {
        mtp_ulong mtp_primitive;        /* always MTP_BIND_REQ */
        mtp_ulong mtp_addr_length;      /* length of MTP address */
        mtp_ulong mtp_addr_offset;      /* offset of MTP address */
        mtp_ulong mtp_bind_flags;       /* bind flags */
} MTP_bind_req_t;
```

Parameters

mtp_primitive:	Is the primitive type. This field is always coded MTP_BIND_REQ.
mtp_addr_length:	Is the length in bytes of the MTP address (MTP-SAPI) to be bound to the stream.
mtp_addr_offset:	Is the offset from the beginning of the M_PROTO block where the MTP address (MTP-SAPI) begins.
mtp_bind_flags:	See "Flags" below.

Flags

Only one of the following flags may be set:

MTP_MANAGEMENT:
When set, this flag indicates that this stream is to be bound only as MTP management and not as an MTP-User. The stream can then invoke MTP Management services for the bound MTP entity.

MTP_CONNECTION_ORIENTED:
When set, this flag specifies that this stream is to be bound for COMS service regardless of the Service Indicator in the bound address.

MTP_CONNECTIONLESS:
When set, this flag specifies that this stream is to be bound for CLMS service regardless of the Service Indicator in the bound address.

When all flags are clear, the COMS or CLMS service type will be chosen according to the Service Indicator in the specified address.

Valid States

This primitive is valid in state MTPS_UNBND (see Appendix B).

New State

The new state is MTPS_WACK_BREQ.

Acknowledgements

The MTP will generate one of the following acknowledgements upon receipt of the MTP_BIND_REQ primitive:

- **Successful**: Correct acknowledgement of the primitive is indicated via the MTP_BIND_ACK primitive.
- **Non-fatal errors**: These errors will be indicated viat the MTP_ERROR_ACK primitive. The applicable non-fatal errors are as follows:

MSYSERR:	A system error occured and the UNIX system error is indicated in the primitive.
MOUTSTATE:	The primitive was issued from an invalid state.
MBADADDR:	The MTP address (MTP-SAPI) was in an incorrect format or the address contained illegal information. It is not intended to indicate protocol errors.
MNOADDR:	The MTP-User did not provide an MTP address (MTP-SAPI) and the MTP could not allocate an address to the user.

MADDRBUSY:	The MTP-User attempted to bind a second stream to an MTP address (MTP-SAPI).
MACCESS:	The MTP-User attempted to bind to an MTP address (MTP-SAPI) for which it did not have sufficient access permissions or attemtped to bind with the MTP_MANAGEMENT flag set and did not have permission to bind as MTP management.
MBOUND:	The MTP-User attempted to bind a second stream to an MTP address with the MTP_MANAGEMENT flag set.
MBADPRIM:	The primitive format was incorrect (i.e. too short).

4.1.6. Bind Protocol Address Acknolwedgement

MTP_BIND_ACK

This primitive indicates to the MTP-User that the specified MTP-User entity has been bound to the requested MTP address (MTP-SAPI).

Format

The format of the meessage is one M_PCPROTO message block, and its structure is the following:

```
typedef struct MTP_bind_ack {
        mtp_ulong mtp_primitive;        /* always MTP_BIND_ACK */
        mtp_ulong mtp_addr_length;      /* length of bound MTP address */
        mtp_ulong mtp_addr_offset;      /* offset of bound MTP address */
} MTP_bind_ack_t;
```

Parameters

mtp_primitive:	Indicates the primitive type. This field is always coded MTP_BIND_ACK.
mtp_addr_length:	Is the length of the MTP address (MTP-SAPI) that was bound.
mtp_addr_offset:	Is the offset from the beginning of the M_PCPROTO block where the MTP address (MTP-SAPI) begins. The proper alignment of the address in the M_PCPROTO message block is not guaranteed.

Rules

The following rules apply to the binding of the specified MTP address to the stream:

- If the *mtp_addr_length* field in the MTP_BIND_REQ primitive is zero, then the MTP is to assign an MTP address (MTP-SAPI) to the user.
- The MTP is to bind the MTP address (MTP-SAPI) as specified in the MTP_BIND_REQ primitive. If the MTP cannot bind the specified address, it may assign another MTP address to the user. It is the MTP-User's responsibility to check the MTP address returned in the MTP_BIND_ACK primitive to see if it is the same as the one requested.

If the above rules result in an error condition, then the MTP must issue an MTP_ERROR_ACK primitive to the MTP-User specifying the error as defined in the description of the MTP_BIND_REQ primitive.

Valid States

This primitive is in response to an MTP_BIND_REQ primitive and is valid in the state MTPS_WACK_BREQ.

New State

The new state is MTPS_IDLE.

4.1.7. Unbind Protocol Address Request

MTP_UNBIND_REQ

This primitive requests that the MTP unbind the MTP-User entity that was previously bound to the MTP address (MTP-SAPI).

Format

The format of the message is one M_PROTO block, and its structure is as follows:

```
typedef struct MTP_unbind_req {
        mtp_ulong mtp_primitive;        /* always MTP_UNBIND_REQ */
} MTP_unbind_req_t;
```

Parameters

mtp_primitive:	Indicates the primitive type. This field is always coded MTP_UNBIND_REQ.

Valid States

This primitive is valid in the MTPS_IDLE state.

New State

The new state is MTPS_WACK_UREQ.

Acknowledgements

This primitive requires the MTP to generate the following acknolwedgements upon receipt of the primitive:

- **Successful**: Correct acknowledgement of the primitive is indicated via the MTP_OK_ACK primtiive.
- **Unsuccessful (Non-fatal errors)**: These errors will be indicated via the MTP_ERROR_ACK primitive. The applicable non-fatal errors are as follows:

MOUTSTATE:	The primitive was issued from an invalid state.
MSYSERR:	A system error has occured and the UNIX system error is indicated in the primtiive.

4.1.8. Message Transfer Part Options Management Request

MTP_OPTMGMT_REQ

This primitive allows the MTP-User to manage the MTP parameter values associated with the stream.

Format

The format of the message is one M_PROTO message block, and its structure is as follows:

```
typedef struct MTP_optmgmt_req {
        mtp_ulong mtp_primitive;        /* always MTP_OPTMGMT_REQ */
        mtp_ulong mtp_opt_length;       /* length of options */
        mtp_ulong mtp_opt_offset;       /* offset of options */
        mtp_ulong mtp_mgmt_flags;       /* management flags */
} MTP_optmgmt_req_t;

#define MTP_DEFAULT     0UL
#define MTP_CHECK       1UL
#define MTP_NEGOTIATE   2UL
#define MTP_CURRENT     3UL
```

Parameters

mtp_primitive:	Specifies the primitive type. This field is always coded MTP_OPTMGMT_REQ.
mtp_opt_length:	Specifies the length of the default values of the options parameters as selected by the MTP-User. These values will be used in subsequent M_DATA transfers to the stream. If the MTP-User cannot determine the value of an option, it value should be set to MTP_UNKNOWN. If the MTP-User does not specify any option parameter values, the length of this field should be set to zero.
mtp_opt_offset:	Specifies the offset of the options parameters from the beginning of the M_PROTO message block.
mtp_mgmt_flags:	See "Flags" below.

Flags

MTP_DEFAULT
: Requests that the MTP return the default option settings for the specified (or all) options in a MTP_OPTMGMT_ACK primitive.

MTP_CHECK
: Requests that the MTP check the specified options and return the success or failure of each specified option in an MTP_OPTMGMT_ACK primitive.

MTP_NEGOTIATE
: Requests that the MTP negotiate the value of the specified options and return the success or failure and the negotiated value in an MTP_OPTMGMT_ACK primitive.

MTP_CURRENT
: Requests that the MTP return the current option settings for the specified (or all) options in a MTP_OPTMGMT_ACK primitive.

Only one of the above flags can be set.

Valid States

This primitive is valid in the MTPS_IDLE state.

New State

The new state is MTPS_WACK_OPTREQ.

Acknowledgements

The MTP_OPTMGMT_REQ primitive requires the MTP to generate one of the following acknowledgements upon receipt of the primitive:

− **Successful**: Acknowledgement is via the MTP_OK_ACK primitive. At successful completions, the resulting state is MTPS_IDLE.

− **Non-fatal errors**: These errors are indicated in the MTP_ERROR_ACK primitive. The resulting state remains unchanged. The applicable non-fatal errors are defined as follows:

MSYSERR:	A system error has occurred and the UNIX system error is indicated in the primitive.
MOUTSTATE:	The primitive was issued from an invalid state.
MBADOPT:	The option parameter values specified are outside the range supported by the MTP.
MBADOPTTYPE:	The option structure tupe is not supported by the MTP.
MBADFLAG:	The flags were invalid or unsupported, or the combination of flags was invalid.
MBADPRIM:	The primitive format was incorrect (i.e. too short).
MACCESS:	The user did not have proper permissions.

4.1.9. Error Acknowledgement

MTP_ERROR_ACK

This primitive indicates to the MTP-User that a non-fatal error has occured in the last MTP-User or MTP-Management originated primitive. This may only be initiated as an acknowledgement for those primitives that require one. It also indicates to the user that no action was taken on the primitive that caused the error.

Format

The format of the mssage is one M_PCPROTO message block, and its structure is as follows:

```
typedef struct MTP_error_ack {
        mtp_ulong mtp_primitive;        /* always MTP_ERROR_ACK */
        mtp_ulong mtp_error_primitive;  /* primitive in error */
        mtp_ulong mtp_mtpi_error;       /* MTP interface error */
        mtp_ulong mtp_unix_error;       /* UNIX error */
} MTP_error_ack_t;
```

Parameters

mtp_primitive:	Identifies the primitive type. This field is always coded MTP_ERROR_ACK.
mtp_error_primitive:	Identifies the primitive type that cause the error.
mtp_mtpi_error:	Contains the Message Transfer Part Interface error code.
mtp_unix_error:	Contains the UNIX system error code. This may only be non-zero if the *mtp_mtpi_error* is equal to MSYSERR.

Valid Error Codes

The following error codes are allowed to be returned:

MSYSERR:	A system error has occurred and the UNIX system error is indicated in the primitive.
MOUTSTATE:	The primitive was issued from an invalid state.
MBADADDR:	The MTP address as specified in the primitive was in an incorrect format, or the address contained illegal information.
MBADOPT:	The options values as specified in the primitive were in an incorrect format, or they contained illegal information.
MBADOPTTYPE:	The option structure tupe is not supported by the MTP.
MNOADDR:	The MTP could not allocate an address.
MADDRBUSY:	The MTP could not use the specified address because the specified address is already in use.
MBADFLAG:	The flags specified in the primitive were incorrect or illegal.
MNOTSUPPORT:	Specified primitive type is not known to the MTP.
MBADPRIM:	The primitive was of an incorrect format (i.e. too small, or an offset it out of range).
MACCESS:	The user did not have proper permissions.

Valid States

This primitive is valid in all states that have a pending acknowledgment or confirmation.

New State

The new state is the same as the one from which the acknowledged request or response was issued.

4.1.10. Successful Receipt Acknowledgements

MTP_OK_ACK

The primitive indicates to the MTP-User that the previous MTP-User or management originated primitive was received successfully by the MTP. It does not indicate to the MTP-User any MTP protocol action taken due to the issuance of the last primitive. The MTP_OK_ACK primitive may only be initiated as an acknowledgement for those user or management originated primitives that have no other means of confirmation.

Format

The format of the mssage is one M_PCPROTO message block, and its structrue is as follows:

```
typedef struct MTP_ok_ack {
        mtp_ulong mtp_primitive;        /* always MTP_OK_ACK */
        mtp_ulong mtp_correct_prim;     /* correct primitive */
} MTP_ok_ack_t;
```

Parameters

mtp_primitive:	Identifies the primitive. This field is always coded MTP_OK_ACK.
mtp_correct_prim:	Identifies the successfully received primitive type.

Valid States

This primitive is issued in states MTPS_WACK_UREQ, MTPS_WACK_OPTREQ, MTPS_WACK_CREQ and MTPS_WACK_DREQ.

New State

The resulting state depends on the current state (see Appendix B, Tables B-7 and B-8.).

4.2. Connection Mode Primitives

This section describes the format of the COMS primitives and the rules associated with these primitives. The default values of the options parameters associated with an MTP association may be selected via the MTP_OPTMGMT_REQ primitive.

4.2.1. Connection Establishment Phase

The following MTP service primitives pertain to the establishment of an association between local and remote MTP-SAPs, provided the MTP users exist, and are known to the MTP.

4.2.1.1. Message Transfer Part Connection Request

MTP_CONN_REQ

This primitive requests that the MTP form an association to the specified destination.

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO message block is as follows:

```
typedef struct MTP_conn_req {
        mtp_ulong mtp_primitive;        /* always MTP_CONN_REQ */
        mtp_ulong mtp_addr_length;      /* length of MTP address to connect */
        mtp_ulong mtp_addr_offset;      /* offset of MTP address to connect */
        mtp_ulong mtp_conn_flags;       /* connect flags */
} MTP_conn_req_t;
```

Parameters

mtp_primitive:	Specifies the primitive type. This field is always coded MTP_CONN_REQ.
mtp_addr_length:	Specifies the length of the MTP address (MTP-SAPI) of the peer to which a signalling relation is to be established. If no MTP address is provided by the MTP-User, this field must be coded zero (0). The coding of the MTP address (MTP-SAPI) is protocol and provider-specific.
mtp_addr_offset:	Specifies the offset of the MTP address (MTP-SAPI) from the beginning of the M_PROTO message block.
mtp_conn_flags:	Indicates a bit field of connection options. (See “Flags” below.)

Flags

Rules

The following rules apply to the establishment of associations between the specified addresses:

- If the *mtp_addr_length* field in the MTP_CONN_REQ primitive is zero, then the MTP is to select a remote MTP address (MTP-SAPI) with which to associate. If the MTP cannot select a address for the association, the MTP responds with an MTP_ERROR_ACK primitive with error MNOADDR.
- If the *mtp_addr_length* field in the MTP_CONN_REQ primitive is non-zero, the MTP is to associate with the specified remote MTP address (MTP-SAPI). if the MTP cannot associate with the specified remote MTP address (MTP-SAPI), the primitive will fail and the MTP will return an MTP_ERROR_ACK primitive with the appropriate error value (e.g, MBADADDR).

The following rules apply to the MTP addresses (MTP-SAPIs):

- If the MTP-User does not specify an MTP address (i.e. *mtp_addr_length* is set to zero), then the MTP may attempt to assign a remote MTP address (MTP-SAPI) and associate it with the stream for the duration of the association.

Valid States

This primitive is valid in state MTPS_IDLE.

New State

The new state is MTPS_DATA_XFER.

Acknowledgements

The following acknolwedgements are valid for this primitive:

- **Successful**: Correct acknowledgement of the primtive is indicated via the MTP_OK_ACK primitive.
- **Unsuccessful (Non-fatal errors)**: These are indicated via the MTP_ERROR_ACK primitive. The applicable non-fatal errors are defined as follows:

MSYSERR:	A system error has occurred and the UNIX system eror is indicated in the primitive.
MOUTSTATE:	The primitive was issued from an invalid state.
MBADADDR:	The MTP address as specified in the primitive was in an incorrect format, or the address contained illegal information.
MNOADDR:	The user did not provide an MTP address and one was required by the MTP. The MTP could not select a remote MTP address (MTP-SAPI).
MADDRBUSY:	The MTP could not use the specified address because the specified address is already in use.
MBADFLAG:	The specified flags were invalid.
MNOTSUPPORT:	The primitive is not supported for by the MTP provider.
MBADPRIM:	The primitive was of an incorrect format (i.e. too small, or an offset it out of range).
MACCESS:	The user did not have proper permissions for the use of the requested address.

4.2.2. Data Transfer Phase

The data transfer service primitive provide for an exchange of MTP-User data, known as MSDUs, in either direction or in both directions simultaneously on a signalling relation. The MTP service preserves the sequence of MSDUs that have the same Signalling Link Selection (SLS) value specified in the MTP_TRANSFER_REQ. MSDUs are self-contained messages with implicit boundaries.

The following MTP service primtiives pertain to the Data Transfer phase of a signalling relation.

4.2.2.1. Message Transfer Part Transfer Request

MTP_TRANSFER_REQ

This user-originated primitive indicates to the MTP that this message contains MTP-User data. It allows the transfer of MTP-User data between MTP-Users, without modification by the MTP provider.

The MTP-User must send an integral number of octets of data greater than zero. In a case where the size of the MSDU exceeds the MIDU (as specified by the size of the MIDU_size parameter of the MTP_INFO_ACK primitive), the MSDU may be broken up into more than one MIDU. When an MSDU is broken up into more than one MIDU, the MTP_MORE_DATA_FLAG will be set on each MIDU except the last one.

Format

The format of the message is one or more M_DATA blocks. Use of a M_PROTO message block is optional. The M_PROTO message block is used for two reasons:

(1) to indicate that the MSDU is broken into more than one MIDU, and that the data carried in the following M_DATA message block constitutes one MIDU;

(2) to indicate the message priority and signalling link selection to be associated with the MSDU.

Guidelines for the use of M_PROTO

The following guidelines must be followed with respect to the use of the M_PROTO message block:

(1) The M_PROTO message block need not be present when the MSDU size is less than or equal to the MIDU size and one of the following is true:

- the values of the message priority and signalling link selection have been previously set with the MTP_OPTMGMT_REQ primitive; or
- the default values of the message priority and signalling link selection are to be specified.

(2) The M_PROTO message block must be present when:

- The MSDU size is greater than the MIDU size.
- the values of the message priority or signalling link selection as specified by the MTP_OPTMGMT_REQ primitive needs to be overriden for this MSDU.

The format of the M_PROTO message block, if present, is as follows:

```
typedef struct MTP_transfer_req {
        mtp_ulong mtp_primitive;        /* always MTP_TRANSFER_REQ */
        mtp_ulong mtp_dest_length;      /* length of destination address */
        mtp_ulong mtp_dest_offset;      /* offset of destination address */
        mtp_ulong mtp_mp;               /* message priority */
        mtp_ulong mtp_sls;              /* signalling link selection */
} MTP_transfer_req_t;
```

Parameters

mtp_primitive:	Specifies the primitive type. This field is always coded MTP_TRANSFER_REQ.
mtp_dest_length:	Specifies the length of the destination MTP address (MTP-SAPI) to which the message is to be sent. If the stream is connected in an established signalling relation, this field may be coded zero (0).
mtp_dest_offset:	Specifies the offset of the destination MTP address (MTP-SAPI) from the beginning of the M_PROTO message block.
mtp_mp:	Specifies the message priority to be used when sending the message. Support for message priority is protocol variant and provider specific.

mtp_sls:	Specifies the signalling link selection value to be used when sending the message. If this field is coded MTP_UNKNOWN (-1) then the MTP provider will choose a value of the signalling link selection which best performs loadsharing of the resulting message traffic.

Rules

If the signalling relation is associated with a single destination address (MTP-SAPI), M_DATA blocks can be used with no M_PROTO block to transfer messages. The destination adddress and the values of SLS and MP will be the default values or the laster values that were successfully set with MTP_OPTMGMT_REQ.

Valid States

This primitive is valid in state MTP_IDLE for connectionless streams and in state MTP_DATA_XFER for pseudo-connection oriented streams.

New State

The new state is unchanged.

Acknowledgements

This primitive does not require any acknowledgements, although it may generate a fatal error. This is indicated to the MTP-User via a M_ERROR STREAMS message type (specifying an errno value of EPROTO) which results in the failure of all system calls on that stream. The applicable errors are defined as follows:

EPROTO: This indicates one of the following unrecoverable protocol conditions:

- The MTP interface was found to be in an incorrect state.
- The amount of MTP-User data associated wtih the primitive was outside the range supported by the MTP (as specified in the MIDU_size parameter of the MTP_INFO_ACK primitive).
- The options requested are either not support by the MTP.
- The M_PROTO message block was not followed by one or more M_DATA message blocks.
- The amount of MTP-User data associated with the current MSDU is outside the range supported by the MTP (as specified by the MSDU_size parameter in the MTP_INFO_ACK primitive.)
- The flags field contained an unknown value.

Note: If the interface is in the MTP_IDLE state when the provider received the MTP_TRANSFER_REQ primitive, then the MTP should discard the request without generating a fatal error.

4.2.2.2. Message Transfer Part Transfer Indication

MTP_TRANSFER_IND

Format

The format of this message is one M_PROTO message block followed by one or more M_DATA blocks. The structure of the M_PROTO block is as follows:

```
typedef struct MTP_transfer_ind {
        mtp_ulong mtp_primitive;        /* always MTP_TRANSFER_IND */
        mtp_ulong mtp_srce_length;      /* length of source address */
        mtp_ulong mtp_srce_offset;      /* offset of source address */
        mtp_ulong mtp_mp;               /* message priority */
        mtp_ulong mtp_sls;              /* signalling link selection */
} MTP_transfer_ind_t;
```

Parameters

mtp_primitive:	Indicates the primitive type. This field is always coded MTP_TRANSFER_IND.
mtp_srce_length:	
mtp_srce_offset:	
mtp_mp:	
mtp_sls:	

Rules

Valid States

This primitive is valid in state MTP_DATA_XFER.

New State

The new state is unchanged.

4.2.2.3. Message Transfer Part Status Indication

MTP_STATUS_IND

Format

The format of the message is one M_PCPROTO message block. The structure of the M_PCPROTO block is as follows:

```
typedef struct MTP_status_ind {
        mtp_ulong mtp_primitive;        /* always MTP_STATUS_IND */
        mtp_ulong mtp_addr_length;      /* length of affected MTP address */
        mtp_ulong mtp_addr_offset;      /* offset of affected MTP address */
        mtp_ulong mtp_type;             /* type */
        mtp_ulong mtp_status;           /* status */
} MTP_status_ind_t;

/* Type for MTP_STATUS_IND */
#define MTP_STATUS_TYPE_CONG            0x00    /* congestion */
#define MTP_STATUS_TYPE_UPU             0x01    /* user part unavailability */

/* Status for MTP_STATUS_IND, with MTP_STATUS_TYPE_UPU */
#define MTP_STATUS_UPU_UNKNOWN          0x01    /* unknown */
#define MTP_STATUS_UPU_UNEQUIPPED       0x02    /* unequipped remote user. */
#define MTP_STATUS_UPU_INACCESSIBLE     0x03    /* inaccessible remote user.
                                                 */

/* Status for MTP_STATUS_IND, with MTP_STATUS_TYPE_CONG */
#define MTP_STATUS_CONGESTION_LEVEL0    0x00    /* congestion level 0 */
#define MTP_STATUS_CONGESTION_LEVEL1    0x01    /* congestion level 1 */
#define MTP_STATUS_CONGESTION_LEVEL2    0x02    /* congestion level 2 */
#define MTP_STATUS_CONGESTION_LEVEL3    0x03    /* congestion level 3 */
#define MTP_STATUS_CONGESTION           0x04    /* congestion */
```

Parameters

mtp_primitive:	Indicates the primitive type. This field is always coded MTP_STATUS_IND.
mtp_addr_length:	Indicates the length of the affected MTP address (MTP-SAPI).
mtp_addr_offset:	Indicates the offset of the affected MTP address (MTP-SAPI) from the beginning of the M_PCPROTO message block.
mtp_type:	Inidcates the type of the status indication. See “Type and Status” below.
mtp_status:	Inidcates the status of the status indication. See “Type and Status” below.

Type and Status

MTP_STATUS_TYPE_CONG:

This type indicates that the affected MTP adddress (MTP-SAPI) is experiencing congestion and that the mtp_status field contains information pertaining to the degree of congestion experienced by the affected destination as follows:

MTP_STATUS_CONGESTION_LEVEL0

Signalling network congestion towards the affected destination has dropped to level zero (0), indicating that there is no longer signalling network congestion towards the affected destination. Whether this indication is given at all is protocol variant and provider-specific.

MTP_STATUS_CONGESTION_LEVEL1

Signalling network congestion towards the affected destination has onset or abated to level one (1). If possible, the MTP-User should withdraw from issuing MTP_TRANSFER_REQ primitives with mtp_mp fields set to less than one (1) to the affected destination.

MTP_STATUS_CONGESTION_LEVEL2

Signalling network congestion towards the affected destination has onset or abated to level two (2). If possible, the MTP-User should withdraw from issuing MTP_TRANSFER_REQ primitives with mtp_mp fields set to less than two (2) to the affected destination.

MTP_STATUS_CONGESTION_LEVEL3

Signalling network congestion towards the affected destination has onset or abated to level three (3). If possible, the MTP-User should withdraw from issuing MTP_TRANSFER_REQ primitives with mtp_mp fields set to less than three (3) to the affected destination.

MTP_STATUS_CONGESTION
: Signalling network congestion exists towards the affected destination. If possible, the MTP-User should withdraw from issuing MTP_TRANSFER_REQ primitives of lower priority to the affected destination.

MTP congestion level status is protocol variant and provider specific. See the Addendum for more information.

MTP_STATUS_TYPE_UPU:
: This type indicates that the affected MTP address (MTP-SAPI) has no accessible corresponding MTP-User and that the mtp_status field contains information pertianing to the nature of the inaccessibility of the remote MTP-User at the affected destination as follows:

MTP_STATUS_UPU_UNKNOWN
: Indicates that the reason for inaccessibility of the remote MTP-User is unknown.

MTP_STATUS_UPU_UNEQUIPPED
: Indicates that the reason for inaccessibility of the remote MTP-User is that the remote MTP-User is not equipped at the affected MTP address (MTP-SAPI).

MTP_STATUS_UPU_INACCESSIBLE
: Indicates that the reason for inaccessibility of the remote MTP-User is that the remote MTP-User is temporarily inaccessible at the affected MTP address (MTP-SAPI).

MTP user part status is protocol variant and provider specific. See the Addendum for more information.

Rules

Valid States

This primitive is valid in state MTPS_DATA_XFER.

New State

The new state is unchanged (MTPS_DATA_XFER).

4.2.2.4. Message Transfer Part Pause Indication

MTP_PAUSE_IND

This primitive indicates to the MTP-User that the indicated remote MTP-entity (signalling point) is temporarily inaccessible. This implies the inaccessibility of remote MTP-User at the affected signalling point.

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

```
typedef struct MTP_pause_ind {
        mtp_ulong mtp_primitive;        /* always MTP_PAUSE_IND */
        mtp_ulong mtp_addr_length;      /* length of affected MTP address */
        mtp_ulong mtp_addr_offset;      /* offset of affected MTP address */
} MTP_pause_ind_t;
```

Parameters

mtp_primitive:	Indicates the primitive type.
mtp_addr_length:	Indicates the length of the MTP address (MTP-SAPI) corresponding to the affected remote MTP-entity.
mtp_addr_offset:	Indicates the offset of the MTP address (MTP-SAPI) from the beginning of the M_PROTO message block.

Rules

Valid States

This primitive is valid in state MTPS_DATA_XFER.

New State

The new state is unchanged (MTPS_DATA_XFER).

4.2.2.5. Message Transfer Part Resume Indication

MTP_RESUME_IND

This primitive indicates to the MTP-User that a previously inaccesible remote MTP-entity (signalling point) is now accessible. This does not imply the accessibility of the remote MTP-User at the affected signalling point: the MTP-User is responsible for sending protocol messages to the remote MTP-User to test its accessibility.

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

```
typedef struct MTP_resume_ind {
        mtp_ulong mtp_primitive;        /* always MTP_RESUME_IND */
        mtp_ulong mtp_addr_length;      /* length of affected MTP address */
        mtp_ulong mtp_addr_offset;      /* offset of affected MTP address */
} MTP_resume_ind_t;
```

Parameters

mtp_primitive:	Indicates the primitive type.
mtp_addr_length:	Indicates the length of the MTP address (MTP-SAPI) corresponding to the affected remote MTP-entity.
mtp_addr_offset:	Indicates the offset of the MTP address (MTP-SAPI) from the beginning of the M_PROTO message block.

Rules

Valid States

This primitive is valid in state MTPS_DATA_XFER.

New State

The new state is unchanged (MTPS_DATA_XFER).

4.2.2.6. Message Transfer Part Restart Complete Indication

MTP_RESTART_COMPLETE_IND

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

```
typedef struct MTP_restart_complete_ind {
        mtp_ulong mtp_primitive;        /* always MTP_RESTART_COMPLETE_IND */
} MTP_restart_complete_ind_t;
```

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

This primitive is valid in state MTPS_DATA_XFER.

New State

The new state is unchanged (MTPS_DATA_XFER).

4.2.3. Signalling Relation Release Phase

4.2.3.1. Message Transfer Part Disconnect Request

MTP_DISCON_REQ

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

```
typedef struct MTP_discon_req {
        mtp_ulong mtp_primitive;        /* always MTP_DISCON_REQ */
} MTP_discon_req_t;
```

Parameters

mtp_primitive:	Specifies the primitive type.

Rules

Valid States

This primitive is valid in state MTPS_DATA_XFER.

New State

The new state is MTPS_WACK_DREQ.

Acknowledgements

4.3. MTP Provider Management Primitives

4.3.1. Link Management Primitives

4.3.1.1. Link Inhibit Request

MTP_INHIBIT_REQ

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Specifies the primitive type.

Valid States

New State

Acknowledgements

4.3.1.2. Link Inhibit Indication

MTP_INHIBIT_IND

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.1.3. Link Inhibit Confirmation

MTP_INHIBIT_CON

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.1.4. Link Uninhibit Request

MTP_UNINHIBIT_REQ

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Specifies the primitive type.

Valid States

New State

Acknowledgements

4.3.1.5. Link Uninhibit Indication

MTP_UNINHIBIT_IND

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.1.6. Link Uninhibit Confirmation

MTP_UNINHIBIT_CON

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.2. Route Management Primitives

4.3.2.1. Route Prohibit Request

MTP_PROHIBIT_REQ

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Specifies the primitive type.

Valid States

New State

Acknowledgements

4.3.2.2. Route Prohibit Indication

MTP_PROHIBIT_IND

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.2.3. Route Prohibit Confirmation

MTP_PROHIBIT_CON

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.2.4. Route Allow Request

MTP_ALLOW_REQ

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Specifies the primitive type.

Valid States

New State

Acknowledgements

4.3.2.5. Route Allow Indication

MTP_ALLOW_IND

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.2.6. Route Allow Confirmation

MTP_ALLOW_CON

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.3. Layer Management Primitives

4.3.3.1. Layer Event Indication

MTP_EVENT_IND

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

4.3.3.2. Statistics Indication

MTP_STATS_IND

Format

The format of the message is one M_PROTO message block. The structure of the M_PROTO block is as follows:

Parameters

mtp_primitive:	Indicates the primitive type.

Rules

Valid States

New State

5. Diagnostics Requirements

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

5.1. Non-Fatal Error Handling Facility

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

5.2. Fatal Error Handling Facility

These errors are issued by the MTP when it detects errors that are not correctable by the MTP service user, or if it is unable to report a correctable error to the MTP service 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 MTP service 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.

6. MTP Management Controls

6.1. MTP Protocol Object Options

Argument Format

```
typedef struct mtp_option {
        ulong type;                             /* object type */
        ulong id;                               /* object id */
        /* followed by object-specific protocol options structure */
} mtp_option_t;
```

Fields

type The object type. The object type is one of the following objects:

MTP_OBJ_TYPE_SL	Signalling link object.
MTP_OBJ_TYPE_LK	Link set object.
MTP_OBJ_TYPE_LS	Combined Link Set object.
MTP_OBJ_TYPE_RT	Route object.
MTP_OBJ_TYPE_RL	Route List object.
MTP_OBJ_TYPE_RS	Route Set object.
MTP_OBJ_TYPE_SP	Signalling Point object.
MTP_OBJ_TYPE_NA	Network Appearance object.
MTP_OBJ_TYPE_DF	Default object.

id The object identifier.

6.1.1. Get MTP Protocol Object Options

MTP_IOCGOPTION

Get the protocol options associated with the identified object.

6.1.2. Set MTP Protocol Object Options

MTP_IOCSOPTION

Set the protocol options associated with the identified object.

6.1.3. Signalling Link Options

```
typedef struct mtp_opt_conf_sl {
        /* signalling link timers */
        ulong t1;                               /* timer t1 value */
        ulong t2;                               /* timer t2 value */
        ulong t3;                               /* timer t3 value */
        ulong t4;                               /* timer t4 value */
        ulong t5;                               /* timer t5 value */
        ulong t12;                              /* timer t12 value */
        ulong t13;                              /* timer t13 value */
        ulong t14;                              /* timer t14 value */
        ulong t17;                              /* timer t17 value */
        ulong t19a;                             /* timer t19a value */
        ulong t20a;                             /* timer t20a value */
        ulong t21a;                             /* timer t21a value */
        ulong t22;                              /* timer t22 value */
        ulong t23;                              /* timer t23 value */
        ulong t24;                              /* timer t24 value */
        ulong t31a;                             /* timer t31a value */
        ulong t32a;                             /* timer t32a value */
        ulong t33a;                             /* timer t33a value */
        ulong t34a;                             /* timer t34a value */
        ulong t1t;                              /* timer t1t value */
        ulong t2t;                              /* timer t2t value */
        ulong t1s;                      /* timer t1s value */
} mtp_opt_conf_sl_t;
```

6.1.4. Link Set Options

```
typedef struct mtp_opt_conf_lk {
        /* signalling link timers */
        ulong t1;                       /* timer t1 value */
        ulong t2;                       /* timer t2 value */
        ulong t3;                       /* timer t3 value */
        ulong t4;                       /* timer t4 value */
        ulong t5;                       /* timer t5 value */
        ulong t12;                      /* timer t12 value */
        ulong t13;                      /* timer t13 value */
        ulong t14;                      /* timer t14 value */
        ulong t17;                      /* timer t17 value */
        ulong t19a;                     /* timer t19a value */
        ulong t20a;                     /* timer t20a value */
        ulong t21a;                     /* timer t21a value */
        ulong t22;                      /* timer t22 value */
        ulong t23;                      /* timer t23 value */
        ulong t24;                      /* timer t24 value */
        ulong t31a;                     /* timer t31a value */
        ulong t32a;                     /* timer t32a value */
        ulong t33a;                     /* timer t33a value */
        ulong t34a;                     /* timer t34a value */
        ulong t1t;                      /* timer t1t value */
        ulong t2t;                      /* timer t2t value */
        ulong t1s;                      /* timer t1s value */
        /* link timers */
        ulong t7;                       /* timer t7 value */
} mtp_opt_conf_lk_t;
```

6.1.5. Combined Link Set Options

```
typedef struct mtp_opt_conf_ls {
        /* signalling link timers */
        ulong t1;                       /* timer t1 value */
        ulong t2;                       /* timer t2 value */
        ulong t3;                       /* timer t3 value */
        ulong t4;                       /* timer t4 value */
        ulong t5;                       /* timer t5 value */
        ulong t12;                      /* timer t12 value */
        ulong t13;                      /* timer t13 value */
        ulong t14;                      /* timer t14 value */
        ulong t17;                      /* timer t17 value */
        ulong t19a;                     /* timer t19a value */
        ulong t20a;                     /* timer t20a value */
        ulong t21a;                     /* timer t21a value */
        ulong t22;                      /* timer t22 value */
        ulong t23;                      /* timer t23 value */
        ulong t24;                      /* timer t24 value */
        ulong t31a;                     /* timer t31a value */
        ulong t32a;                     /* timer t32a value */
        ulong t33a;                     /* timer t33a value */
        ulong t34a;                     /* timer t34a value */
        ulong t1t;                      /* timer t1t value */
        ulong t2t;                      /* timer t2t value */
        ulong t1s;                      /* timer t1s value */
        /* link timers */
        ulong t7;                       /* timer t7 value */
} mtp_opt_conf_ls_t;
```

6.1.6. Route Options

```
typedef struct mtp_opt_conf_rt {
        /* route timers */
        ulong t6;                       /* timer t6 value */
        ulong t10;                      /* timer t10 value */
} mtp_opt_conf_rt_t;
```

6.1.7. Route List Options

```
typedef struct mtp_opt_conf_rl {
        /* route timers */
        ulong t6;                               /* timer t6 value */
        ulong t10;                              /* timer t10 value */
} mtp_opt_conf_rl_t;
```

6.1.8. Route Set Options

```
typedef struct mtp_opt_conf_rs {
        /* route timers */
        ulong t6;                               /* timer t6 value */
        ulong t10;                              /* timer t10 value */
        /* route set timers */
        ulong t8;                               /* timer t8 value */
        ulong t11;                              /* timer t11 value */
        ulong t15;                              /* timer t15 value */
        ulong t16;                              /* timer t16 value */
        ulong t18a;                             /* timer t18a value */
} mtp_opt_conf_rs_t;
```

6.1.9. Signalling Point Options

```
typedef struct mtp_opt_conf_sp {
        /* signalling link timers */
        ulong t1;                               /* timer t1 value */
        ulong t2;                               /* timer t2 value */
        ulong t3;                               /* timer t3 value */
        ulong t4;                               /* timer t4 value */
        ulong t5;                               /* timer t5 value */
        ulong t12;                              /* timer t12 value */
        ulong t13;                              /* timer t13 value */
        ulong t14;                              /* timer t14 value */
        ulong t17;                              /* timer t17 value */
        ulong t19a;                             /* timer t19a value */
        ulong t20a;                             /* timer t20a value */
        ulong t21a;                             /* timer t21a value */
        ulong t22;                              /* timer t22 value */
        ulong t23;                              /* timer t23 value */
        ulong t24;                              /* timer t24 value */
        ulong t31a;                             /* timer t31a value */
        ulong t32a;                             /* timer t32a value */
        ulong t33a;                             /* timer t33a value */
        ulong t34a;                             /* timer t34a value */
        ulong t1t;                              /* timer t1t value */
        ulong t2t;                              /* timer t2t value */
        ulong t1s;                              /* timer t1s value */
        /* link timers */
        ulong t7;                               /* timer t7 value */
        /* route timers */
        ulong t6;                               /* timer t6 value */
        ulong t10;                              /* timer t10 value */
        /* route set timers */
        ulong t8;                               /* timer t8 value */
        ulong t11;                              /* timer t11 value */
        ulong t15;                              /* timer t15 value */
        ulong t16;                              /* timer t16 value */
        ulong t18a;                             /* timer t18a value */
        /* signalling point timers */
        ulong t1r;                              /* timer t1r value */
        ulong t18;                              /* timer t18 value */
        ulong t19;                              /* timer t19 value */
        ulong t20;                              /* timer t20 value */
        ulong t21;                              /* timer t21 value */
        ulong t22a;                             /* timer t22a value */
        ulong t23a;                             /* timer t23a value */
        ulong t24a;                             /* timer t24a value */
        ulong t25a;                             /* timer t25a value */
        ulong t26a;                             /* timer t26a value */
        ulong t27a;                             /* timer t27a value */
        ulong t28a;                         /* timer t28a value */
        ulong t29a;                         /* timer t29a value */
        ulong t30a;                         /* timer t30a value */
} mtp_opt_conf_sp_t;
```

6.1.10. Network Appearance Options

```
typedef struct mtp_opt_conf_na {
        /* signalling link timers */
        ulong t1;                           /* timer t1 value */
        ulong t2;                           /* timer t2 value */
        ulong t3;                           /* timer t3 value */
        ulong t4;                           /* timer t4 value */
        ulong t5;                           /* timer t5 value */
        ulong t12;                          /* timer t12 value */
        ulong t13;                          /* timer t13 value */
        ulong t14;                          /* timer t14 value */
        ulong t17;                          /* timer t17 value */
        ulong t19a;                         /* timer t19a value */
        ulong t20a;                         /* timer t20a value */
        ulong t21a;                         /* timer t21a value */
        ulong t22;                          /* timer t22 value */
        ulong t23;                          /* timer t23 value */
        ulong t24;                          /* timer t24 value */
        ulong t31a;                         /* timer t31a value */
        ulong t32a;                         /* timer t32a value */
        ulong t33a;                         /* timer t33a value */
        ulong t34a;                         /* timer t34a value */
        ulong t1t;                          /* timer t1t value */
        ulong t2t;                          /* timer t2t value */
        ulong t1s;                          /* timer t1s value */
        /* link timers */
        ulong t7;                           /* timer t7 value */
        /* route timers */
        ulong t6;                           /* timer t6 value */
        ulong t10;                          /* timer t10 value */
        /* route set timers */
        ulong t8;                           /* timer t8 value */
        ulong t11;                          /* timer t11 value */
        ulong t15;                          /* timer t15 value */
        ulong t16;                          /* timer t16 value */
        ulong t18a;                         /* timer t18a value */
        /* signalling point timers */
        ulong t1r;                          /* timer t1r value */
        ulong t18;                          /* timer t18 value */
        ulong t19;                          /* timer t19 value */
        ulong t20;                          /* timer t20 value */
        ulong t21;                          /* timer t21 value */
        ulong t22a;                         /* timer t22a value */
        ulong t23a;                         /* timer t23a value */
        ulong t24a;                         /* timer t24a value */
        ulong t25a;                         /* timer t25a value */
        ulong t26a;                         /* timer t26a value */
        ulong t27a;                         /* timer t27a value */
        ulong t28a;                         /* timer t28a value */
        ulong t29a;                         /* timer t29a value */
        ulong t30a;                         /* timer t30a value */
} mtp_opt_conf_na_t;
```

6.1.11. Default Options

```
typedef struct mtp_opt_conf_df {
} mtp_opt_conf_df_t;
```

6.2. MTP Protocol Object Configuration

Argument Format

```
typedef struct mtp_config {
        ulong type;                             /* object type */
        ulong id;                               /* object id */
        ulong cmd;                              /* configuration command */
        /* followed by object-specific configuration structure */
} mtp_config_t;

#define MTP_GET         0       /* get configuration */
#define MTP_ADD         1       /* add configuration */
#define MTP_CHA         2       /* cha configuration */
#define MTP_DEL         3       /* del configuration */
```

type	The object type. The object type is one of the following objects:

`MTP_OBJ_TYPE_SL`	Signalling link object.
`MTP_OBJ_TYPE_LK`	Link set object.
`MTP_OBJ_TYPE_LS`	Combined Link Set object.
`MTP_OBJ_TYPE_RT`	Route object.
`MTP_OBJ_TYPE_RL`	Route List object.
`MTP_OBJ_TYPE_RS`	Route Set object.
`MTP_OBJ_TYPE_SP`	Signalling Point object.
`MTP_OBJ_TYPE_NA`	Network Appearance object.
`MTP_OBJ_TYPE_DF`	Default object.

id	The object identifier.
cmd	The configuration command to execute. The command may be one of the following:

`MTP_GET`	Get the specified protocol object configuration including the configuration of as many direct descendants of the object as possible.
`MTP_ADD`	Add the specified protocol object.
`MTP_CHA`	Change the specified protocol object.
`MTP_DEL`	Delete the specified protocol object.

6.2.1. Get MTP Protocol Object Configuration

MTP_IOCGCONFIG

6.2.2. Set MTP Protocol Object Configuration

MTP_IOCSCONFIG

6.2.3. Test MTP Protocol Object Configuration

MTP_IOCTCONFIG

6.2.4. Commit MTP Protocol Object Configuration

MTP_IOCCCONFIG

6.2.5. Signalling Link Configuration

```
typedef struct mtp_conf_sl {
        ulong muxid;                            /* lower multiplexor id */
        ulong lkid;                             /* link set id */
        ulong slc;                              /* signalling link code in lk */
} mtp_conf_sl_t;
```

6.2.6. Link Set Configuration

```
typedef struct mtp_conf_lk {
        ulong lsid;                     /* combined link set id */
        ulong rsid;                     /* routeset of adjacent signalling point */
        ulong ni;                       /* network indicator for link set */
        ulong slot;                     /* slot of SLS for this link set */
} mtp_conf_lk_t;
```

6.2.7. Combined Link Set Configuration

```
typedef struct mtp_conf_ls {
        ulong spid;                     /* signalling point id */
        ulong sls_mask;                 /* mask of bits selecting link set */
} mtp_conf_ls_t;
```

6.2.8. Route Configuration

```
typedef struct mtp_conf_rt {
        ulong rlid;                     /* route list id */
        ulong lkid;                     /* link id */
        ulong slot;                     /* slot of SLS for this route */
} mtp_conf_rt_t;
```

6.2.9. Route List Configuration

```
typedef struct mtp_conf_rl {
        ulong rsid;                     /* route set id */
        ulong lsid;                     /* combined link set id */
        ulong cost;                     /* cost in routeset */
} mtp_conf_rl_t;
```

6.2.10. Route Set Configuration

```
typedef struct mtp_conf_rs {
        ulong spid;                     /* signalling point id */
        ulong dest;                     /* destination point code */
        ulong flags;                    /* options flags */
} mtp_conf_rs_t;
```

6.2.11. Signalling Point Configuration

```
typedef struct mtp_conf_sp {
        ulong naid;                     /* network appearance id */
        ulong pc;                       /* point code */
        ulong users;                    /* mask of equipped users */
        ulong flags;                    /* options flags */
} mtp_conf_sp_t;
```

6.2.12. Network Appearance Configuration

```
typedef struct mtp_conf_na {
        lmi_option_t options;           /* protocol options */
        struct {
                ulong member;           /* PC member mask */
                ulong cluster;          /* PC cluster mask */
                ulong network;          /* PC network mask */
        } mask;
        ulong sls_bits;                 /* bits in SLS */
} mtp_conf_na_t;

/* additional MTP protocol options */
#define SS7_POPT_TFR    0x00010000      /* old broadcast method - no responsive */
#define SS7_POPT_TFRB   0x00020000      /* new broadcast method - no regulation */
#define SS7_POPT_TFRR   0x00040000      /* new responsive method - regulated */
#define SS7_POPT_MCSTA  0x00080000      /* multiple congestion states */
```

6.2.13. Default Configuration

```
typedef struct mtp_conf_df {
} mtp_conf_df_t;
```

6.3. MTP Protocol Object State Machine

Argument Format

```
typedef struct mtp_statem {
        ulong type;                         /* object type */
        ulong id;                           /* object id */
        ulong flags;                        /* object flags */
        ulong state;                        /* object state */
        /* followed by object-specific state structure */
} mtp_statem_t;
```

6.3.1. Signalling Link State

```
typedef struct mtp_timers_sl {
        ulong t1;                           /* timer t1 */
        ulong t2;                           /* timer t2 */
        ulong t3;                           /* timer t3 */
        ulong t4;                           /* timer t4 */
        ulong t5;                           /* timer t5 */
        ulong t12;                          /* timer t12 */
        ulong t13;                          /* timer t13 */
        ulong t14;                          /* timer t14 */
        ulong t17;                          /* timer t17 */
        ulong t19a;                         /* timer t19a */
        ulong t20a;                         /* timer t20a */
        ulong t21a;                         /* timer t21a */
        ulong t22;                          /* timer t22 */
        ulong t23;                          /* timer t23 */
        ulong t24;                          /* timer t24 */
        ulong t31a;                         /* timer t31a */
        ulong t32a;                         /* timer t32a */
        ulong t33a;                         /* timer t33a */
        ulong t34a;                         /* timer t34a */
        ulong t1t;                          /* timer t1t */
        ulong t2t;                          /* timer t2t */
        ulong t1s;                          /* timer t1s */
} mtp_timers_sl_t;
typedef struct mtp_statem_sl {
        struct mtp_timers_sl timers;
} mtp_statem_sl_t;

#define SLS_OUT_OF_SERVICE      0       /* out of service */
#define SLS_PROC_OUTG           1       /* processor outage */
#define SLS_IN_SERVICE          2       /* in service */
#define SLS_WACK_COO            3       /* waiting COA/ECA in response to COO */
#define SLS_WACK_ECO            4       /* waiting COA/ECA in response to ECO */
#define SLS_WCON_RET            5       /* waiting for retrieval confrimation */
#define SLS_WIND_CLRB           6       /* waiting for clear buffers indication */
#define SLS_WIND_BSNT           7       /* waiting for BSNT indication */
#define SLS_WIND_INSI           8       /* waiting for in service indication */
#define SLS_WACK_SLTM           9       /* waiting SLTA in response to 1st SLTM */

#define SL_RESTORED          (MTP_ALLOWED)       /* Sig link Activated/Restored/Resumed */
#define SL_DANGER            (MTP_DANGER)        /* Sig link Danger of congestion (overloaded) */
#define SL_CONGESTED         (MTP_CONGESTED)     /* Sig link Congested (link congestion) */
#define SL_UNUSABLE          (MTP_RESTRICTED)    /* Sig link Unusable (Local Processor Outage) */
#define SL_RETRIEVAL         (MTP_RESTART)       /* Sig link Retrieving */
#define SL_FAILED            (MTP_PROHIBITED)    /* Sig link Failed */
#define SL_INHIBITED         (MTP_INHIBITED)     /* Sig link Inhibited (Management inhibited) */
#define SL_BLOCKED           (MTP_BLOCKED)       /* Sig link Blocked (Processor Outage) */
#define SL_INACTIVE          (MTP_INACTIVE)      /* Sig link Inactive (Out of Service) */
#define SL_NODANGER          (MTP_NODANGER)      /* Sig link Out of Danger (transient state) */
#define SL_UNCONGESTED       (MTP_UNCONGESTED)   /* Sig link Uncongested (transient state) */
#define SL_UPDATED           (MTP_RESTARTED)     /* Sig link Buffer Update Complete (transient state) */
#define SL_UNINHIBITED       (MTP_UNINHIBITED)   /* Sig link Uninhibited (transient state) */
#define SL_UNBLOCKED         (MTP_UNBLOCKED)     /* Sig link Unblocked (transient state) */
#define SL_ACTIVE            (MTP_ACTIVE)        /* Sig link Active (Link in service) */

#define SLF_TRAFFIC          (MTPF_TRAFFIC)      /* Sig link has sent traffic */
#define SLF_COO_RECV         (MTPF_COO_RECV)     /* Sig link has received a COO */
#define SLF_ECO_RECV         (MTPF_ECO_RECV)     /* Sig link has received a ECO */
```

```
#define SLF_WACK_SLTM       (MTPF_WACK_SLTM)    /* Sig link waiting for response to 1st SLTM */
#define SLF_WACK_SLTM2      (MTPF_WACK_SLTM2)   /* Sig link waiting for response to 2nd SLTM */
#define SLF_WACK_SSLTM      (MTPF_WACK_SSLTM)   /* Sig link waiting for response to 1st SSLTM */
#define SLF_WACK_SSLTM2     (MTPF_WACK_SSLTM2)  /* Sig link waiting for response to 2nd SSLTM */

#define SLF_RESTORED        (MTPF_ALLOWED)      /* Sig link Activated/Restored */
#define SLF_DANGER          (MTPF_DANGER)       /* Sig link Danger of congestion (overloaded) */
#define SLF_CONGESTED       (MTPF_CONGESTED)    /* Sig link Congested (link congestion) */
#define SLF_UNUSABLE        (MTPF_RESTRICTED)   /* Sig link Unusable (Local Processor Outage) */
#define SLF_RETRIEVAL       (MTPF_RESTART)      /* Sig link Retrieving */
#define SLF_FAILED          (MTPF_PROHIBITED)   /* Sig link Failed */
#define SLF_INHIBITED       (MTPF_INHIBITED)    /* Sig link Inhibited (Management inhibited) */
#define SLF_BLOCKED         (MTPF_BLOCKED)      /* Sig link Blocked (Processor Outage) */
#define SLF_INACTIVE        (MTPF_INACTIVE)     /* Sig link Inactive (Out of Service) */

#define SLF_LOSC_PROC_A     (MTPF_LOSC_PROC_A)  /* Sig link uses link oscillation procedure A */
#define SLF_LOSC_PROC_B     (MTPF_LOSC_PROC_B)  /* Sig link uses link oscillation procedure B */
```

6.3.2. Link Set State

```
typedef struct mtp_timers_lk {
        ulong t7;                               /* timer t7 */
} mtp_timers_lk_t;
typedef struct mtp_statem_lk {
        struct mtp_timers_lk timers;
} mtp_statem_lk_t;

#define LK_ALLOWED          (MTP_ALLOWED)       /* Link Allowed */
#define LK_DANGER           (MTP_DANGER)        /* Link Danger of congestion (primary or secondary) */
#define LK_CONGESTED        (MTP_CONGESTED)     /* Link Congested (Link Set congestion, primary or secondary ) */
#define LK_RESTRICTED       (MTP_RESTRICTED)    /* Link Restricted (Route Failure or received TFR) */
#define LK_RESTART          (MTP_RESTART)       /* Link Restarting */
#define LK_PROHIBITED       (MTP_PROHIBITED)    /* Link Prohibited (Received TFP) */
#define LK_INHIBITED        (MTP_INHIBITED)     /* Link Inhibited (Management inhibited) */
#define LK_BLOCKED          (MTP_BLOCKED)       /* Link Blocked (Local Link Set failure) */
#define LK_INACTIVE         (MTP_INACTIVE)      /* Link Inactive (Link out of service) */
#define LK_NODANGER         (MTP_NODANGER)      /* Link Out of Danger (transient state) */
#define LK_UNCONGESTED      (MTP_UNCONGESTED)   /* Link Uncongested (transient state) */
#define LK_RESTARTED        (MTP_RESTARTED)     /* Link Restarted */
#define LK_UNINHIBITED      (MTP_UNINHIBITED)   /* Link Uninhibited (transient state) */
#define LK_UNBLOCKED        (MTP_UNBLOCKED)     /* Link Unblocked (transient state) */
#define LK_ACTIVE           (MTP_ACTIVE)        /* Link Active (Link in service) */
```

6.3.3. Combined Link Set State

```
typedef struct mtp_timers_ls {
} mtp_timers_ls_t;
typedef struct mtp_statem_ls {
        struct mtp_timers_ls timers;
} mtp_statem_ls_t;

#define LS_ALLOWED          (MTP_ALLOWED)       /* Linkset Allowed */
#define LS_DANGER           (MTP_DANGER)        /* Linkset Danger of congestion (primary or secondary) */
#define LS_CONGESTED        (MTP_CONGESTED)     /* Linkset Congested (Link Set congestion, primary or secondary )
                                                 */
#define LS_RESTRICTED       (MTP_RESTRICTED)    /* Linkset Restricted (Route Failure or received TFR) */
#define LS_RESTART          (MTP_RESTART)       /* Linkset Restarting */
#define LS_PROHIBITED       (MTP_PROHIBITED)    /* Linkset Prohibited (Received TFP) */
#define LS_INHIBITED        (MTP_INHIBITED)     /* Linkset Inhibited (Management inhibited) */
#define LS_BLOCKED          (MTP_BLOCKED)       /* Linkset Blocked (Local Link Set failure) */
#define LS_INACTIVE         (MTP_INACTIVE)      /* Linkset Inactive (Link out of service) */
#define LS_NODANGER         (MTP_NODANGER)      /* Linkset Out of Danger (transient state) */
#define LS_UNCONGESTED      (MTP_UNCONGESTED)   /* Linkset Uncongested (transient state) */
#define LS_RESTARTED        (MTP_RESTARTED)     /* Linkset Restarted */
#define LS_UNINHIBITED      (MTP_UNINHIBITED)   /* Linkset Uninhibited (transient state) */
#define LS_UNBLOCKED        (MTP_UNBLOCKED)     /* Linkset Unblocked (transient state) */
#define LS_ACTIVE           (MTP_ACTIVE)        /* Linkset Active (Link in service) */
```

6.3.4. Route State

```
typedef struct mtp_timers_rt {
        ulong t6;                               /* timer t6 */
        ulong t10;                              /* timer t10 */
} mtp_timers_rt_t;
typedef struct mtp_statem_rt {
        struct mtp_timers_rt timers;
} mtp_statem_rt_t;

#define RT_ALLOWED          (MTP_ALLOWED)       /* Route Allowed */
#define RT_DANGER           (MTP_DANGER)        /* Route Danger of congestion (primary or secondary) */
#define RT_CONGESTED        (MTP_CONGESTED)     /* Route Congested (Link Set congestion, primary or secondary ) *
#define RT_RESTRICTED       (MTP_RESTRICTED)    /* Route Restricted (Route Failure or received TFR) */
#define RT_RESTART          (MTP_RESTART)       /* Route Restarting */
#define RT_PROHIBITED       (MTP_PROHIBITED)    /* Route Prohibited (Received TFP) */
#define RT_INHIBITED        (MTP_INHIBITED)     /* Route Inhibited (Management inhibited) */
#define RT_BLOCKED          (MTP_BLOCKED)       /* Route Blocked (Local Link Set failure) */
#define RT_INACTIVE         (MTP_INACTIVE)      /* Route Inactive (Link out of service) */
#define RT_NODANGER         (MTP_NODANGER)      /* Route Out of Danger (transient state) */
#define RT_UNCONGESTED      (MTP_UNCONGESTED)   /* Route Uncongested (transient state) */
#define RT_RESTARTED        (MTP_RESTARTED)     /* Route Restarted */
#define RT_UNINHIBITED      (MTP_UNINHIBITED)   /* Route Uninhibited (transient state) */
#define RT_UNBLOCKED        (MTP_UNBLOCKED)     /* Route Unblocked (transient state) */
#define RT_ACTIVE           (MTP_ACTIVE)        /* Route Active (Link in service) */
//#define RT_RESTART_PHASE_1  (MTP_RESTART_PHASE_1)     /* Route Restarting Phase 1 */
//#define RT_RESTART_PHASE_2  (MTP_RESTART_PHASE_2)     /* Route Restarting Phase 2 */

#define RTF_ALLOWED         (MTPF_ALLOWED)      /* Route is allowed */
#define RTF_DANGER          (MTPF_DANGER)       /* Route is in danger of congestion */
#define RTF_CONGESTED       (MTPF_CONGESTED)    /* Route is congested */
#define RTF_RESTRICTED      (MTPF_RESTRICTED)   /* Route is restricted */
#define RTF_RESTART         (MTPF_RESTART)      /* Route is restarting */
#define RTF_PROHIBITED      (MTPF_PROHIBITED)   /* Route is prohibited */
#define RTF_INHIBITED       (MTPF_INHIBITED)    /* Route is inhibited */
#define RTF_BLOCKED         (MTPF_BLOCKED)      /* Route is blocked */
#define RTF_INACTIVE        (MTPF_INACTIVE)     /* Route is inactive */
```

6.3.5. Route List State

```
typedef struct mtp_timers_rl {
} mtp_timers_rl_t;
typedef struct mtp_statem_rl {
        struct mtp_timers_rl timers;
} mtp_statem_rl_t;

//#define RL_ALLOWED        (MTP_ALLOWED)       /* Routelist Allowed */
//#define RL_DANGER         (MTP_DANGER)        /* Routelist Danger of congestion (primary or secondary) */
//#define RL_CONGESTED      (MTP_CONGESTED)     /* Routelist Congested (Link Set cong, primary or secondary) */
#define RL_RESTRICTED       (MTP_RESTRICTED)    /* Routelist Restricted (Route Failure or received TFR) */
#define RL_RESTART          (MTP_RESTART)       /* Routelist Restarting */
//#define RL_PROHIBITED     (MTP_PROHIBITED)    /* Routelist Prohibited (Received TFP) */
//#define RL_INHIBITED      (MTP_INHIBITED)     /* Routelist Inhibited (Management inhibited) */
//#define RL_BLOCKED        (MTP_BLOCKED)       /* Routelist Blocked (Local Link Set failure) */
//#define RL_INACTIVE       (MTP_INACTIVE)      /* Routelist Inactive (Link out of service) */
//#define RL_NODANGER       (MTP_NODANGER)      /* Routelist Out of Danger (transient state) */
//#define RL_UNCONGESTED            (MTP_UNCONGESTED)   /* Routelist Uncongested (transient state) */
//#define RL_RESTARTED      (MTP_RESTARTED)     /* Routelist Restarted */
//#define RL_UNINHIBITED            (MTP_UNINHIBITED)   /* Routelist Uninhibited (transient state) */
//#define RL_UNBLOCKED      (MTP_UNBLOCKED)     /* Routelist Unblocked (transient state) */
//#define RL_ACTIVE         (MTP_ACTIVE)        /* Routelist Active (Link in service) */
//#define RL_RESTART_PHASE_1  (MTP_RESTART_PHASE_1)     /* Routelist Restarting Phase 1 */
//#define RL_RESTART_PHASE_2  (MTP_RESTART_PHASE_2)     /* Routelist Restarting Phase 2 */
```

6.3.6. Route Set State

```
typedef struct mtp_timers_rs {
        ulong t8;                               /* timer t8 */
        ulong t11;                              /* timer t11 */
        ulong t15;                              /* timer t15 */
        ulong t16;                              /* timer t16 */
        ulong t18a;                           /* timer t18a */
} mtp_timers_rs_t;
typedef struct mtp_statem_rs {
        struct mtp_timers_rs timers;
} mtp_statem_rs_t;

#define RS_ALLOWED          (MTP_ALLOWED)       /* Routeset Allowed */
#define RS_DANGER           (MTP_DANGER)        /* Routeset Danger of congestion (primary or secondary) */
#define RS_CONGESTED        (MTP_CONGESTED)     /* Routeset Congested (Link Set cong, primary or secondary) */
#define RS_RESTRICTED       (MTP_RESTRICTED)    /* Routeset Restricted (Route Failure or received TFR) */
#define RS_RESTART          (MTP_RESTART)       /* Routeset Restarting */
#define RS_PROHIBITED       (MTP_PROHIBITED)    /* Routeset Prohibited (Received TFP) */
#define RS_INHIBITED        (MTP_INHIBITED)     /* Routeset Inhibited (Management inhibited) */
#define RS_BLOCKED          (MTP_BLOCKED)       /* Routeset Blocked (Local Link Set failure) */
#define RS_INACTIVE         (MTP_INACTIVE)      /* Routeset Inactive (Link out of service) */
#define RS_NODANGER         (MTP_NODANGER)      /* Routeset Out of Danger (transient state) */
#define RS_UNCONGESTED      (MTP_UNCONGESTED)   /* Routeset Uncongested (transient state) */
#define RS_RESTARTED        (MTP_RESTARTED)     /* Routeset Restarted */
#define RS_UNINHIBITED      (MTP_UNINHIBITED)   /* Routeset Uninhibited (transient state) */
#define RS_UNBLOCKED        (MTP_UNBLOCKED)     /* Routeset Unblocked (transient state) */
#define RS_ACTIVE           (MTP_ACTIVE)        /* Routeset Active (Link in service) */
#define RS_RESTART_PHASE_1  (MTP_RESTART_PHASE_1)       /* Routeset Restarting Phase 1 */
#define RS_RESTART_PHASE_2  (MTP_RESTART_PHASE_2)       /* Routeset Restarting Phase 2 */

#define RSF_ALLOWED         (MTPF_ALLOWED)      /* Routeset is allowed */
#define RSF_DANGER          (MTPF_DANGER)       /* Routeset is in danger of congestion */
#define RSF_CONGESTED       (MTPF_CONGESTED)    /* Routeset is congested */
#define RSF_RESTRICTED      (MTPF_RESTRICTED)   /* Routeset is restricted */
#define RSF_RESTART         (MTPF_RESTART)      /* Routeset is restarting */
#define RSF_PROHIBITED      (MTPF_PROHIBITED)   /* Routeset is prohibited */
#define RSF_INHIBITED       (MTPF_INHIBITED)    /* Routeset is inhibited */
#define RSF_BLOCKED         (MTPF_BLOCKED)      /* Routeset is blocked */
#define RSF_INACTIVE        (MTPF_INACTIVE)     /* Routeset is inactive */

#define RSF_TFR_PENDING     (MTPF_TFR_PENDING)  /* Routeset has TFR pending */
#define RSF_CLUSTER         (MTPF_CLUSTER)      /* Routeset is cluster route */
#define RSF_XFER_FUNC       (MTPF_XFER_FUNC)    /* Routeset has transfer function */
#define RSF_ADJACENT        (MTPF_ADJACENT)     /* Routeset is adjacent */
```

6.3.7. Signalling Point State

```
typedef struct mtp_timers_sp {
        ulong t1r;                            /* timer t1r */
        ulong t18;                            /* timer t18 */
        ulong t19;                            /* timer t19 */
        ulong t20;                            /* timer t20 */
        ulong t21;                            /* timer t21 */
        ulong t22a;                           /* timer t22a */
        ulong t23a;                           /* timer t23a */
        ulong t24a;                           /* timer t24a */
        ulong t25a;                           /* timer t25a */
        ulong t26a;                           /* timer t26a */
        ulong t27a;                           /* timer t27a */
        ulong t28a;                           /* timer t28a */
        ulong t29a;                           /* timer t29a */
        ulong t30a;                           /* timer t30a */
} mtp_timers_sp_t;
typedef struct mtp_statem_sp {
        struct mtp_timers_sp timers;
} mtp_statem_sp_t;

#define SP_ALLOWED          (MTP_ALLOWED)       /* Sig Point Allowed */
#define SP_DANGER           (MTP_DANGER)        /* Sig Point Danger of congestion (primary or secondary) */
#define SP_CONGESTED        (MTP_CONGESTED)     /* Sig Point Congested (Link Set cong, primary or secondary ) */
#define SP_RESTRICTED       (MTP_RESTRICTED)    /* Sig Point Restricted (Route Failure or received TFR) */
#define SP_RESTART          (MTP_RESTART)       /* Sig Point Restarting */
#define SP_PROHIBITED       (MTP_PROHIBITED)    /* Sig Point Prohibited (Received TFP) */
#define SP_INHIBITED        (MTP_INHIBITED)     /* Sig Point Inhibited (Management inhibited) */
#define SP_BLOCKED          (MTP_BLOCKED)       /* Sig Point Blocked (Local Link Set failure) */
#define SP_INACTIVE         (MTP_INACTIVE)      /* Sig Point Inactive (Link out of service) */
#define SP_NODANGER         (MTP_NODANGER)      /* Sig Point Out of Danger (transient state) */
#define SP_UNCONGESTED      (MTP_UNCONGESTED)   /* Sig Point Uncongested (transient state) */
```

```
#define SP_RESTARTED        (MTP_RESTARTED)     /* Sig Point Restarted */
#define SP_UNINHIBITED      (MTP_UNINHIBITED)   /* Sig Point Uninhibited (transient state) */
#define SP_UNBLOCKED        (MTP_UNBLOCKED)     /* Sig Point Unblocked (transient state) */
#define SP_ACTIVE           (MTP_ACTIVE)        /* Sig Point Active (Link in service) */
#define SP_RESTART_PHASE_1  (MTP_RESTART_PHASE_1)       /* Sig Point Restarting Phase 1 */
#define SP_RESTART_PHASE_2  (MTP_RESTART_PHASE_2)       /* Sig Point Restarting Phase 2 */

#define SPF_RESTART         (MTPF_RESTART)      /* Sig Point restarting */
#define SPF_CLUSTER         (MTPF_CLUSTER)      /* Sig Point is cluster route */
#define SPF_XFER_FUNC       (MTPF_XFER_FUNC)    /* Sig Point has transfer function */
#define SPF_SECURITY        (MTPF_SECURITY)     /* Sig Point has additional security procedures */
#define SPF_LOSC_PROC_A     (MTPF_LOSC_PROC_A)  /* Sig Point uses link oscillation procedure A */
#define SPF_LOSC_PROC_B     (MTPF_LOSC_PROC_B)  /* Sig Point uses link oscillation procedure B */
#define SPF_RESTART_PHASE_1 (MTPF_RESTART_PHASE_1)      /* Sig Point restarting */
#define SPF_RESTART_PHASE_2 (MTPF_RESTART_PHASE_2)      /* Sig Point restarting */
```

6.3.8. Network Appearance State

```
typedef struct mtp_timers_na {
} mtp_timers_na_t;
typedef struct mtp_statem_na {
        struct mtp_timers_na timers;
} mtp_statem_na_t;
```

6.3.9. Default State

```
typedef struct mtp_timers_df {
} mtp_timers_df_t;
typedef struct mtp_statem_df {
        struct mtp_timers_df timers;
} mtp_statem_df_t;
```

6.3.10. Get MTP Protocol Object State Machine

MTP_IOCGSTATEM

6.3.11. Reset MTP Protocol Object State Machine

MTP_IOCCMRESET

6.4. MTP Protocol Object Statistics

Argument Format

```
typedef struct mtp_stats {
        ulong type;                             /* object type */
        ulong id;                               /* object id */
        ulong header;                           /* object stats header */
        /* followed by object-specific statistics structure */
} mtp_stats_t;
```

6.4.1. Get MTP Protocol Object Statistics Periods

MTP_IOCGSTATSP

6.4.2. Set MTP Protocol Object Statistics Periods

MTP_IOCSSTATSP

6.4.3. Get MTP Protocol Object Statistics

MTP_IOCGSTATS

6.4.4. Set MTP Protocol Object Statistics

MTP_IOCSSTATS

6.5. MTP Protoocl Object Notifications

Argument Format

```
typedef struct mtp_notify {
        ulong type;                             /* object type */
        ulong id;                               /* object id */
        /* followed by object-specific notification structure */
} mtp_notify_t;
```

6.5.1. Get MTP Protocol Object Notifications

MTP_IOCGNOTIFY

6.5.2. Set MTP Protocol Object Notifications

MTP_IOCSNOTIFY

6.5.3. Clear MTP Protocol Object Notifications

MTP_IOCCNOTIFY

6.6. MTP Protocol Object Management

Argument Format

```
typedef struct mtp_mgmt {
        ulong type;                          /* object type */
        ulong id;                            /* object id */
        ulong cmd;                           /* mgmt command */
} mtp_mgmt_t;

#define MTP_MGMT_ALLOW                 0
#define MTP_MGMT_RESTRICT              1
#define MTP_MGMT_PROHIBIT              2
#define MTP_MGMT_ACTIVATE              3
#define MTP_MGMT_DEACTIVATE            4
#define MTP_MGMT_BLOCK                 5
#define MTP_MGMT_UNBLOCK               6
#define MTP_MGMT_INHIBIT               7
#define MTP_MGMT_UNINHIBIT             8
#define MTP_MGMT_CONGEST               9
#define MTP_MGMT_UNCONGEST            10
#define MTP_MGMT_DANGER               11
#define MTP_MGMT_NODANGER             12
#define MTP_MGMT_RESTART              13
#define MTP_MGMT_RESTARTED            14
```

6.6.1. Manage MTP Protocol Object

MTP_IOCCMGMT

6.7. MTP Provider Pass-Along Control

Argument Format

```
typedef struct mtp_pass {
        ulong muxid;                        /* mux index of lower SL structure to pass message to */
        ulong type;                         /* type of message block */
        ulong band;                         /* band of mesage block */
        ulong ctl_length;                   /* length of cntl part */
        ulong dat_length;                   /* length of data part */
        /* followed by cntl and data part of message to pass to signalling link */
} mtp_pass_t;
```

MTP_IOCCPASS

a. Addendum for Q.931 Conformance

This addendum describes the formats and rules that are specific to ISDN Q.931. The addendum must be used along with the generic MTPI as defined in the main document when implementing an MTP that will be configured with the Q.931 call processing layer.

a.1. Primitives and Rules for Q.931 Conformance

The following are the rules that apply to the MTPI primitives for Q.931 compatibility.

a.1.1. Common Primitive Parameters

a.1.1.1. Message Transfer Part Addresses

Format

The format of the MTP address is as follows:

Parameters

mtp_flags:	Indicates the flags associated with the primitive. See "Flags" below.
mtp_addr_length:	Indicates the length of the address.
mtp_addr_offset:	Indicates the offset of the address from the beginning of the block.

Flags

In primitive in which the flags and address parameters both occur, the flag parameter can have the following flag bits set:

MTP_CHANNEL	Indicates that the address is interpreted by the MTP as a channel identifier.
MTP_CHANNEL_GROUP	Indicates that the address is interpreted by the MTP as a channel group identifier.
MTP_TRUNK_GROUP	Indicates that the address is interpreted by the MTP as a trunk group identifier.

Rules

Rules for flags:

(1) In primitives in which the flags and addresss parameters both occur, the flag parameter flag settings indicate the form of the address parameter.

(2) In primitives in which the address parameter occurs without the flags parameter, the form of the address parameter is implied by the primitive.

(3) Only one of MTP_CIRCUIT, MTP_CIRCUIT_GROUP or MTP_TRUNK_GROUP can be specified.

(4) Although a flags parameter exists in conjunction with the address parameter in the primitive, some primitive might not support all flag settings (particularly the MTP_CIRCUIT_GROUP flag). See the particular primitive in this addendum.

Rules for address:

(5) The address contained in the primitive must be one of the following:

- A circuit identifier.
- A circuit group identifier.
- A trunk group identifier.

a.1.1.2. Optional Information Elements

Format

The format of the optional information elements is as follows:

Parameters

mtp_opt_length:	Indicates the length of the optional information elements associated with the primitive. For Q.931 conforming MTP providers, the format of the optional information elements is the format of a Information Element list as specified in Q.931.
mtp_opt_offset:	indicates the offest of the option information elements from the beginning of the block.

Rules

Rules for optional information elements:

(1) The optional information elements provided by the MTP-User may be checked for syntax by the MTP. If the MTP discovers a syntax error in the format of the optional information elements, the MTP should respond with an MTP_ERROR_ACK primitive with error MBADOPT.

(2) For some primitives, specific optional information elements might be interpreted by the MTP and alter the function of some primitives. See the specific primitive descriptions later in this addendum.

(3) Except for optional information elements interpreted by the MTP as specified later in this addendum, the optional information elements are treated as opaque and the optional information element list only is checked for syntax. Opaque information elements will be passed to the ISDN message without examination by the MTP.

(4) To perform specific functions, additional optional information elements may be added to ISDN messages by the MTP.

a.1.2. Local Management Primitives

a.1.2.1. MTP_INFO_ACK

Parameters

Flags

Rules

a.1.2.2. MTP_BIND_REQ

Parameters

mtp_addr_length:	Specifies the length of the address to bind.
mtp_addr_offset:	Specifies the offset of the address to bind.
mtp_setup_ind:	Specifies the requested maximum number of setup indications that will be outstanding for the listening stream.

Flags

MTP_DEFAULT_LISTENER

MTP_CHANNEL

MTP_CHANNEL_GROUP

MTP_TRUNK_GROUP
When on of these flags are set, it indicates that the address is interpreted by the MTP as unspecified (MTP_DEFAULT_LISTENER), a channel (MTP_CHANNEL), as a channel group (MTP_CHANNEL_GROUP), or as a trunk group (MTP_TRUNK_GROUP).

Rules

Rules for address specification:

(1) The addtress contained in the primitive must be either a unspecified, a channel, a channel group or a trunk group.

(2) If the MTP_DEFAULT_LISTENER flag is set, the address should be left unspecified by the MTP-User and should be ignored by the MTP.

Rules for setup indicatesion:

(1) If the number of setup indications is non-zero, the stream is bound as a listening stream. Listening streams will receive all calls that are incoming on the address bound:

- If the address bound is a channel (MTP_CHANNEL flag set), all incoming calls on the channel will be delivered to the stream listening on the channel. These streams will have a maximum number of setup indications of one (1).
- If the address bound is a channel group (MTP_CHANNEL_GROUP flag set), all incoming calls on the channel group will be delivered to the stream listening on the channel group. These streams will have a maximum number of setup indications no higher than the number of channels in the channel group.
- If the address bound is a trunk group (MTP_TRUNK_GROUP flag set), all incoming calls on the trunk group will be delivered to the stream listening on the trunk group. These streams will have a maximum number of setup indications no higher than the number of channels in the trunk group.

Rules for bind flags:

(1) For Q.931 conforming MTP providers, the MTP_DEFAULT_LISTENER will receive incoming calls that have no other listening stream. There can only be one stream bound with the MTP_DEFAULT_LISTENER flag set.

(2) Only one of MTP_DEFAULT_LISTENER, MTP_CHANNEL, MTP_CHANNEL_GROUP or MTP_TRUNK_GROUP may be set.

a.1.2.3. MTP_BIND_ACK

Parameters

Flags

Rules

a.1.2.4. MTP_OPTMGMT_REQ

Parameters

Flags

Rules

a.1.3. Call Setup Primitives

a.1.3.1. Call Type and Flags

Call type and flags are used in the folloiwng primitives:

MTP_SETUP_REQ and MTP_SETUP_IND.

Parameters

mtp_call_type:	Indicates the type of call to be set up. For Q.931 conforming MTP providers, the call type can be one of the call types listed under "Call Type" below.
mtp_call_flags:	Specifies the options flags associated with the call. For Q.931 conforming MTP providers, the call flags can be any of the flags listed under "Flags" below.

Call Type

The following call types are defined for Q.931 conforming MTP providers:

MTP_CALL_TYPE_SPEECH
: The call type is speech. This call type corresponds to a Q.931 Information transfer capability of Speech, and an Information transfer rate of 64kbit/s.

MTP_CALL_TYPE_64KBS_UNRESTRICTED
: The call type is 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information

transfer capability of Unrestricted, and an Information transfer rate of 64kbit/s.

MTP_CALL_TYPE_3_1kHZ_AUDIO
: The call type is 3.1 kHz audio. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of 64kbits/s.

MTP_CALL_TYPE_128KBS_UNRESTRICTED
: The call type is 2 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of 2x64 kbit/s.

MTP_CALL_TYPE_384KBS_UNRESTRICTED
: The call type is 384 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of 384 kbit/s.

MTP_CALL_TYPE_1536KBS_UNRESTRICTED
: The call type is 1536 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of 1536 kbit/s.

MTP_CALL_TYPE_1920KBS_UNRESTRICTED
: The call type is 1920 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of 1920 kbit/s.

MTP_CALL_TYPE_2x64KBS_UNRESTRICTED
: The call type is 2 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 2.

MTP_CALL_TYPE_3x64KBS_UNRESTRICTED
: The call type is 3 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 3.

MTP_CALL_TYPE_4x64KBS_UNRESTRICTED
: The call type is 4 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 4.

MTP_CALL_TYPE_5x64KBS_UNRESTRICTED
: The call type is 5 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 5.

MTP_CALL_TYPE_6x64KBS_UNRESTRICTED
: The call type is 6 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 6.

MTP_CALL_TYPE_7x64KBS_UNRESTRICTED
: The call type is 7 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 7.

MTP_CALL_TYPE_8x64KBS_UNRESTRICTED
: The call type is 8 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 8.

MTP_CALL_TYPE_9x64KBS_UNRESTRICTED
: The call type is 9 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 9.

MTP_CALL_TYPE_10x64KBS_UNRESTRICTED
: The call type is 10 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 10.

MTP_CALL_TYPE_11x64KBS_UNRESTRICTED
: The call type is 11 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 11.

MTP_CALL_TYPE_12x64KBS_UNRESTRICTED
: The call type is 12 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 12.

MTP_CALL_TYPE_13x64KBS_UNRESTRICTED
: The call type is 13 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 13.

MTP_CALL_TYPE_14x64KBS_UNRESTRICTED
: The call type is 14 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 14.

MTP_CALL_TYPE_15x64KBS_UNRESTRICTED
: The call type is 15 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 15.

MTP_CALL_TYPE_16x64KBS_UNRESTRICTED
: The call type is 16 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 16.

MTP_CALL_TYPE_17x64KBS_UNRESTRICTED
: The call type is 17 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 17.

MTP_CALL_TYPE_18x64KBS_UNRESTRICTED
: The call type is 18 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 18.

MTP_CALL_TYPE_19x64KBS_UNRESTRICTED
: The call type is 19 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 19.

MTP_CALL_TYPE_20x64KBS_UNRESTRICTED
: The call type is 20 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 20.

MTP_CALL_TYPE_21x64KBS_UNRESTRICTED
: The call type is 21 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 21.

MTP_CALL_TYPE_22x64KBS_UNRESTRICTED
: The call type is 22 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 22.

MTP_CALL_TYPE_23x64KBS_UNRESTRICTED
: The call type is 23 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 23.

MTP_CALL_TYPE_24x64KBS_UNRESTRICTED
: The call type is 24 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information

transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 24.

MTP_CALL_TYPE_25x64KBS_UNRESTRICTED
: The call type is 25 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 25.

MTP_CALL_TYPE_26x64KBS_UNRESTRICTED
: The call type is 26 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 26.

MTP_CALL_TYPE_27x64KBS_UNRESTRICTED
: The call type is 27 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 27.

MTP_CALL_TYPE_28x64KBS_UNRESTRICTED
: The call type is 28 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 28.

MTP_CALL_TYPE_29x64KBS_UNRESTRICTED
: The call type is 29 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 29.

MTP_CALL_TYPE_30x64KBS_UNRESTRICTED
: The call type is 30 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.931 Information transfer capability of Unrestricted, and an Information transfer rate of multirate with a base rate of 64 kbit/s and a multiplier of 30.

Flags

The following call flags are defined for Q.931 conforming MTP providers:

MTP_ITC_WITH_TONES_AND_ANNOUNCEMENTS
: When set, this flag indicates that the unrestricted digital information includes tones and announcements.

Rules

a.1.3.2. MTP_SETUP_REQ

Parameters

mtp_call_type:	Specifies the type of call to be set up. For Q.931 conforming MTP providers, the call type can be one of the call types listed under "Call Type and Flags" in this addendeum.
mtp_call_flags:	Specifies the options flags associated with the call. For Q.931 conforming MTP providers, the call flags can be any of the flags listed under "Call Type and Flags" in this addendum.
mtp_cdpn_length:	Specifies the length of the called party number. For Q.931 conforming MTP providers, the format of the called party number is the format of the Called Party Number parameter (without the parameter type or length octets) as specified in Q.931.
mtp_cdpn_offset:	Specifies the offset of the called party number from the beginning of the block.

Rules

Rules for call type:

(1) A MTP need not support all of the call types listed.

Rules for call flags:

(1) The MTP_ITC_WITH_TONES_AND_ANNOUNCEMENTS flag may only be set when the call type is unrestricted digital information. When the call type is not unrestricted digital information, this flag should be ignored by the MTP provider.

Rules for called party number:

Rules for generating a SETUP message:

(1) The mandatory (first) Bearer Capability information element in the SETUP message will be derived from the call type and flags. The Bearer Capability information element will contain the Information transfer capability, rate, base and multiplier indicated above.

- When the call type is MTP_CALL_TYPE_128KBS_UNRESTRICTED, the Bearer Capability information element will be coded with an Information transfer capability of unrestricted (or unrestricted with tones an announcements if the flag MTP_ITC_WITH_TONES_AND_ANNOUNCEMENTS i set) and an Information transfer rate of 2 x 64 kbit/s unirate call. For a multirate call, the call type should be coded as MTP_CALL_TYPE_2x64KBS_UNRESTRICTED.
- When the call type is MTP_CALL_TYPE_384KBS_UNRESTRICTED, the Bearer Capability information element will be coded with an Information transfer capability of unrestricted (or unrestricted with tones an announcements if the flag MTP_ITC_WITH_TONES_AND_ANNOUNCEMENTS i set) and an Information transfer rate of 384 kbit/s unirate call. For a multirate call, the call type should be coded as MTP_CALL_TYPE_6x64KBS_UNRESTRICTED.
- When the call type is MTP_CALL_TYPE_1536KBS_UNRESTRICTED, the Bearer Capability information element will be coded with an Information transfer capability of unrestricted (or unrestricted with tones an announcements if the flag MTP_ITC_WITH_TONES_AND_ANNOUNCEMENTS i set) and an Information transfer rate of 1536 kbit/s unirate call. For a multirate call, the call type should be coded as MTP_CALL_TYPE_24x64KBS_UNRESTRICTED.
- When the call type is MTP_CALL_TYPE_1920KBS_UNRESTRICTED, the Bearer Capability information element will be coded with an Information transfer capability of unrestricted (or unrestricted with tones an announcements if the flag MTP_ITC_WITH_TONES_AND_ANNOUNCEMENTS i set) and an Information transfer rate of 1920 kbit/s unirate call. For a multirate call, the call type should be coded as MTP_CALL_TYPE_29x64KBS_UNRESTRICTED.

(2) The mandatory Channel Identification information element in the SETUP message will be derived from the address to which the stream is bound.

- If the stream is bound to a channel group (the one or more interfaces), then a free channel will be selected automatically by the MTP (if possible).
- If the stream is bound to a channel, then the channel identifier of the bound channel will be used.

Rules for state transitions:

(1) If the optional information element contains a Sending Complete information element, then the MTP will not accept any subsequent MTP_INFORMATION_REQ primitives from the MTP-User, nor will the MTP issue any subsequent MTP_MORE_INFO_IND primitives to the MTP-User.

a.1.3.3. MTP_SETUP_IND

Parameters

mtp_call_type:	Specifies the type of call to be set up. For Q.931 conforming MTP providers, the call type can be one of the call types listed under "Call Type and Flags" in this addendum.
mtp_call_flags:	Specifies the options flags associated with the call. For Q.931 conforming MTP providers, the call flags can be any of the flags listed under "Call Type and Flags" in this addendum.
mtp_cdpn_length:	Specifies the length of the called party number. For Q.931 conforming MTP providers, the format of the called party number is the format of the Called Party Number parameter (without the parameter type or length octets) as specified in Q.931.
mtp_cdpn_offset:	Specifies the offset of the called party number from the beginning of the block.
mtp_addr_length:	Specifies the length of the address which contains the channel identifier selected for the call.
mtp_addr_offset:	Specifies the ofsset fo the address from the beginning of the block.

Flags

Call flags can be any of the call flags supported by the MTP listed under MTP_SETUP_REQ in this addendum.

Rules

Rules for call type:

(1) A MTP need not support all of the call types listed.

Rules for call flags:

(1) The MTP_ITC_WITH_TONES_AND_ANNOUNCEMENTS flag may only be set when the call type is unrestricted digital information. When the call type is not unrestricted digital information, this flag should be ignored by the MTP provider.

Rules for called party number:

Rules for obtaining parameters from a SETUP message:

(1) The mandatory (first) Bearer Capability information element in the SETUP message will be translated into the call type and flags. The call type and flags correspond to the Bearer Capability information element will contain the Information transfer capability, rate, base and multiplier indicated under "Call Type" and "Flags".

(2) The mandatory Channel Identification information element in the SETUP message will be provided in the address parameter.

Rules for state transitions:

(1) If the optional information element contains a Sending Complete information element, then the MTP will not accept any subsequent MTP_MORE_INFO_REQ primitives from the MTP-User, nor will the MTP issue any subsequent MTP_INFORMATION_IND primitives to the MTP-User.

a.1.3.4. MTP_SETUP_RES

Parameters

Flags

Rules

a.1.3.5. MTP_SETUP_CON

Parameters

Flags

Rules

a.1.3.6. MTP_CALL_REATTEMPT_IND

Rules

a.1.3.7. MTP_SETUP_COMPLETE_REQ

Parameters

Flags

Rules

a.1.3.8. MTP_SETUP_COMPLETE_IND

Parameters

Flags

Rules

a.1.4. Continuity Check Primitives

a.1.4.1. MTP_CONT_CHECK_REQ

Parameters

Flags

Rules

a.1.4.2. MTP_CONT_CHECK_RES

Parameters

Flags

Rules

a.1.4.3. MTP_CONT_REPORT_REQ

Parameters

Flags

Rules

a.1.5. Call Establishment Primitives

a.1.5.1. MTP_MORE_INFO_REQ

Parameters

Flags

Rules

a.1.5.2. MTP_MORE_INFO_IND

Parameters

Flags

Rules

a.1.5.3. MTP_INFORMATION_REQ

Parameters

Flags

Rules

a.1.5.4. MTP_INFORMATION_IND

Parameters

Flags

Rules

a.1.5.5. MTP_INFO_TIMEOUT_IND

Rules

Rules for issuing primitive:

(1) If the Q.931 conforming MTP is expecting additional digits (it has previously issued an MTP_MORE_INFO_REQ) and timer T302 expires, the MTP will issue this primitive to the MTP-User.

(2) Upon receipt of this primitive, it is the MTP-User's responsibility to determine whether the address digits are sufficient and to issue a MTP_SETUP_RES or MTP_REJECT_REQ primitive.

For compatibility between MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if the MTP-User receives an MTP_INFO_TIMEOUT_IND

a.1.5.6. MTP_PROCEEDING_REQ

Parameters

Flags

Rules

a.1.5.7. MTP_PROCEEDING_IND

Parameters

Flags

Rules

a.1.5.8. MTP_ALERTING_REQ

Parameters

Flags

Rules

a.1.5.9. MTP_ALERTING_IND

Parameters

Flags

Rules

a.1.5.10. MTP_PROGRESS_REQ

Parameters

Flags

Rules

a.1.5.11. MTP_PROGRESS_IND

Parameters

Flags

Rules

a.1.5.12. MTP_IBI_REQ

Parameters

Flags

Rules

a.1.5.13. MTP_IBI_IND

Parameters

Flags

Rules

a.1.6. Call Established Primitives

a.1.6.1. MTP_SUSPEND_REQ

Parameters

mtp_flags:	Indicates the options associated with the suspend. See "Flags" below.

Flags

Q.931 conforming MTP providers do not support suspend flags. This field should be coded zero (0) by the MTP-User and ignored by the MTP.

Rules

Rules for issuing primitive:

(1) Only the MTP-User on the User side of the Q.931 interface can issue a MTP_SUSPEND_REQ primitive. If the MTP is in Network mode and it receives a MTPS_SUSPEND_REQ, it should respond with an MTP_ERROR_ACK with error MNOTSUPPORT.

a.1.6.2. MTP_SUSPEND_IND

mtp_flags:	Indicates the options associated with the suspend. See "Flags" below.

Flags

Q.931 conforming MTP providers do not support suspend flags. This field will be coded zero (0) by the MTP and may be ignored by the MTP.

a.1.6.3. MTP_SUSPEND_RES

Parameters

Rules

a.1.6.4. MTP_SUSPEND_CON

Parameters

Rules

a.1.6.5. MTP_SUSPEND_REJECT_REQ

Parameters

mtp_cause:	Specifies the cause for the rejection. For Q.931 conforming MTP providers, the cause values can be any of the values listed in "Cause Values" in this addendum with the exception of MTPS_CAUS_NONE.

Flags

Rules

a.1.6.6. MTP_SUSPEND_REJECT_IND

Parameters

mtp_cause:	Specifies the cause for the rejection. For Q.931 conforming MTP providers, the cause values can be any of the values listed in "Cause Values" in this addendum with the exception of MTPS_CAUS_NONE.

Flags

Rules

a.1.6.7. MTP_RESUME_REQ

Parameters

mtp_flags:	Indicates the options associated with the resume. See "Flags" below.

Flags

Q.931 conforming MTP providers do not support resume flags. This field should be coded zero (0) by the MTP-User and ignored by the MTP.

Rules

a.1.6.8. MTP_RESUME_IND

Parameters

mtp_flags:	Indicates the options associated with the resume. See "Flags" below.

Flags

Q.931 conforming MTP providers do not support resume flags. This field should be coded zero (0) by the MTP-User and ignored by the MTP.

Rules

a.1.6.9. MTP_RESUME_RES

Parameters

Flags

Rules

a.1.6.10. MTP_RESUME_CON

Parameters

Flags

Rules

a.1.6.11. MTP_RESUME_REJECT_REQ

Parameters

mtp_cause:	Specifies the cause for the rejection. For Q.931 conforming MTP providers, the cause values can be any of the values listed in "Cause Values" in this addendum with the exception of MTPS_CAUS_NONE.

Flags

Rules

a.1.6.12. MTP_RESUME_REJECT_IND

mtp_cause:	Specifies the cause for the rejection. For Q.931 conforming MTP providers, the cause values can be any of the values listed in "Cause Values" in this addendum with the exception of MTPS_CAUS_NONE.

Parameters

Flags

Rules

a.1.7. Call Termination Primitives

a.1.7.1. Cause Values

Cause values are used in the following primitives:

MTP_REJECT_REQ,
MTP_REJECT_IND,
MTP_DISCONNECT_REQ,
MTP_DISCONNECT_IND,
MTP_RELEASE_REQ, and
MTP_RELEASE_IND.

Parameters

mtp_cause:	Indicates the case for the rejection, disconnection, or release of a call. For Q.931 conforming MTP providers, the cause values can be any of the cause values listed in Q.850 listed under "Cause Value" below.

Cause Value

Cause values are essentially opaque and cause values will be transferred directly to the corresponding Q.931 message. The following cause values are defined for Q.931 conforming MTP providers:

MTP_CAUS_UNALLOCATED_NUMBER
: The called party number does not correspond to number allocated to a subscriber or terminal.

MTP_CAUS_NO_ROUTE_TO_TRANSIT_NETWORK

MTP_CAUS_NO_ROUTE_TO_DESTINATION

MTP_CAUS_SEND_SPECIAL_INFO_TONE

MTP_CAUS_MISDIALLED_TRUNK_PREFIX

MTP_CAUS_PREEMPTION

MTP_CAUS_PREEMPTION_CCT_RESERVED

MTP_CAUS_NORMAL_CALL_CLEARING

MTP_CAUS_USER_BUSY

MTP_CAUS_NO_USER_RESPONDING

MTP_CAUS_NO_ANSWER

MTP_CAUS_SUBSCRIBER_ABSENT

MTP_CAUS_CALL_REJECTED

MTP_CAUS_NUMBER_CHANGED

MTP_CAUS_REDIRECT

MTP_CAUS_OUT_OF_ORDER

MTP_CAUS_ADDRESS_INCOMPLETE

MTP_CAUS_FACILITY_REJECTED

MTP_CAUS_NORMAL_UNSPECIFIED

MTP_CAUS_NO_CCT_AVAILABLE

MTP_CAUS_NETWORK_OUT_OF_ORDER

MTP_CAUS_TEMPORARY_FAILURE

MTP_CAUS_SWITCHING_EQUIP_CONGESTION

MTP_CAUS_ACCESS_INFO_DISCARDED

MTP_CAUS_REQUESTED_CCT_UNAVAILABLE

MTP_CAUS_PRECEDENCE_CALL_BLOCKED

MTP_CAUS_RESOURCE_UNAVAILABLE

MTP_CAUS_NOT_SUBSCRIBED

MTP_CAUS_OGC_BARRED_WITHIN_CUG

MTP_CAUS_ICC_BARRED WITHIN_CUG

MTP_CAUS_BC_NOT_AUTHORIZED

MTP_CAUS_BC_NOT_AVAILABLE

MTP_CAUS_INCONSISTENCY

MTP_CAUS_SERVICE_OPTION_NOT_AVAILABLE

MTP_CAUS_BC_NOT_IMPLEMENTED

MTP_CAUS_FACILITY_NOT_IMPLEMENTED

MTP_CAUS_RESTRICTED_BC_ONLY

MTP_CAUS_SERIVCE_OPTION_NOT_IMPLEMENTED

MTP_CAUS_USER_NOT_MEMBER_OF_CUG

MTP_CAUS_INCOMPATIBLE_DESTINATION

MTP_CAUS_NON_EXISTENT_CUG

MTP_CAUS_INVALID_TRANSIT_NTWK_SELECTION

MTP_CAUS_INVALID_MESSAGE

MTP_CAUS_MESSAGE_TYPE_NOT_IMPLEMENTED

MTP_CAUS_PARAMETER_NOT_IMPLEMENTED

MTP_CAUS_RECOVERY_ON_TIMER_EXPIRY

MTP_CAUS_PARAMETER_PASSED_ON

MTP_CAUS_MESSAGE_DISCARDED

MTP_CAUS_PROTOCOL_ERROR

MTP_CAUS_INTERWORKING

MTP_CAUS_UNALLOCATED_DEST_NUMBER

MTP_CAUS_UNKNOWN_BUSINESS_GROUP

MTP_CAUS_EXCHANGE_ROUTING_ERROR

MTP_CAUS_MISROUTED_CALL_TO_PORTED_NUMBER 26

MTP_CAUS_LNP_QOR_NUMBER_NOT_FOUND

MTP_CAUS_PREEMPTION

MTP_CAUS_PRECEDENCE_CALL_BLOCKED

MTP_CAUS_CALL_TYPE_INCOMPATIBLE

MTP_CAUS_GROUP_RESTRICTIONS

Rules

Also to these cause values, the MTP might support additional variant-specific cause values.

a.1.7.2. MTP_REJECT_REQ

Parameters

mtp_cause:	Specifies the cause value for the rejection. For Q.931 conforming MTP providers, the cause value will be one of the cause values listed under "Cause Values" in this Addendum.

Flags

Rules

a.1.7.3. MTP_REJECT_IND

Parameters

mtp_cause:	Specifies the cause value for the rejection. For Q.931 conforming MTP providers, the cause value will be one of the cause values listed under "Cause Values" in this Addendum.

Flags

Rules

a.1.7.4. MTP_CALL_FAILURE_IND

Parameters

mtp_reason:	Specifies the reason for the failure indication. For Q.931 conforming MTP providers, the reason will be one of the reasons listed under "Call Failure Reasons" below.
mtp_cause:	Specifies the cause value for the error indication. For Q.931 conforming MTP providers, the cause value will be one of the cause values listed under "Cause Values" in this addendum.

Call Failure Reasons

MTP_FAILURE_ERROR
: Indicates that the data link failed and recovered during overlap sending or overlap receiving.

MTP_FAILURE_STATUS
: Indicates that the MTP received a STATUS message from the peer with a unrecoverable mismatch in state.

MTP_FAILURE_RESTART
: Indicates that the MTP received or issued a RESTART message for the channel.

Flags

Rules

a.1.7.5. MTP_DISCONNECT_REQ

Parameters

mtp_cause:	Specifies the cause value for the disconnect. For Q.931 conforming MTP providers, the cause value will be one of the cause values listed under "Cause Values" in this addendum.

Rules

a.1.7.6. MTP_DISCONNECT_IND

Parameters

mtp_cause:	Indicates the case values for the disconnect. For Q.931 conforming MTP providers, the cause value wil be one of the cause values listed under "Cause Value" in this addendum.

Rules

a.1.7.7. MTP_RELEASE_REQ

Parameters

mtp_cause:	Specifies the cause value for the release. For Q.931 conforming MTP providers, the cause value will be one of the cause values listed under "Cause Values" in this addendum.

Rules

Rules for cause:

(1) If the request is not the first step in the clearing phase (i.e, the call is not in state MTP_WREQ_REL), then the cause value must be specified. Otherwise, the cause value should be coded MTP_CAUS_NONE by the MTP-User and ignored by the MTP.

a.1.7.8. MTP_RELEASE_IND

Parameters

mtp_cause:	Specifies the cause value for the release. For Q.931 conforming MTP providers, the cause value will be one of the cause values listed under "Cause Values" in this addendum.

Rules

Rules for cause:

(1) If the request is not the first step in the clearing phase (i.e, the call is not in state MTP_WIND_REL), then the cause value will be indicated by the MTP provider. Otherwise, the cause value will be coded MTP_CAUS_NONE by the MTP provider and should be ignored by the MTP-User.

a.1.7.9. MTP_RELEASE_RES

Parameters

Rules

a.1.7.10. MTP_RELEASE_CON

Parameters

Rules

a.1.8. Management Primitives

a.1.8.1. MTP_RESTART_REQ

Parameters

mtp_flags:	
mtp_addr_length:	Specifies the length of the address which contains the interface identifier(s) and optional channel identification for the interface(s) or channels to restart.
mtp_addr_offset:	Specifies the offset of the address from the beginning of the block.

Flags

Rules

a.1.8.2. MTP_RESTART_CON

Parameters

mtp_flags:	
mtp_addr_length:	Specifies the length of the address which contains the interface identifier(s) and optional channel identification for the interface(s) or channels to restart.
mtp_addr_offset:	Specifies the offset of the address from the beginning of the block.

Flags

Rules

b. Addendum for Q.764 Conformance

This addendum describes the formats and rules that are specific to ISUP Q.764. The addendum must be used along with the generic MTPI as defined in the main document when implementing an MTP that will be configured with the Q.764 call processing layer.

b.1. Primitives and Rules for Q.764 Conformace

The following are the rules that apply to the MTPI primitives for Q.764 compatibility.

b.1.1. Common Primitive Parameters

b.1.1.1. Message Transfer Part Addresses

Format

The format of the MTP addresses for Q.764 conforming MTP providers is as follows:

Parameters

mtp_flags:	Indicates the flags associated with the primitive. See "Flags" below.
mtp_addr_length:	Indicates the length of the address.
mtp_addr_offset:	Indicates the offset of the address from the beginning of the block.

Flags

In primitive in which the flags and address parameters both occur, the flag parameter can have the following flag bits set:

MTP_CIRCUIT	Indicates that the address is interpreted by the MTP as a circuit identifier.
MTP_CIRCUIT_GROUP	Indicates that the address is interpreted by the MTP as a circuit group identifier.
MTP_TRUNK_GROUP	Indicates that the address is interpreted by the MTP as a trunk group identifier.

Rules

Rules for flags:

(1) In primitives in which the flags and addresss parameters both occur, the flag parameter flag settings indicate the form of the address parameter.

(2) In primitives in which the address parameter occurs without the flags parameter, the form of the address parameter is implied by the primitive.

(3) Only one of MTP_CIRCUIT, MTP_CIRCUIT_GROUP or MTP_TRUNK_GROUP can be specified.

(4) Although a flags parameter exists in conjunction with the address parameter in the primitive, some primitive might not support all flag settings (particularly the MTP_CIRCUIT_GROUP flag). See the particular primitive in this addendum.

Rules for address:

(5) The address contained in the primitive must be one of the following:

- A circuit identifier.
- A circuit group identifier.
- A trunk group identifier.

b.1.1.2. Optional Parameters

Format

The format of the optional paraemters for Q.764 conforming MTP providers is as follows:

Parameters

mtp_opt_length:	Specifies or indicates the length of the optional parameters associated with the primitive. For Q.764 conforming MTP providers, the format of the optional parameters is the format of the Optional Parameters list (without the pointer or End of Optional Parameters octets) as specified in Q.763.
mtp_opt_offset:	Specifies the offset of the optional parameters from the beginning of the block.

Rules

Rules for optional parameters:

(1) The optional parameters provided by the MTP-User may be checked for syntax by the MTP. If the MTP discovers a syntax error in the format of the optional parameters, the MTP should respond with a MTP_ERROR_ACK primitive with error MBADOPT.

(2) For some primitives, specific optional parameters might be interpreted by the MTP and alter the function of some primitives. See the specific primitive descriptions later in this addendum.

(3) Except for optional parameters interpreted by the MTP as specified later in this addendum, the optional parameters are treated as opaque and the optional parameter list only is checked for syntax. Opaque parameters will be passed to the ISUP message without examination by the MTP.

(4) To perform specific functions, additional optional parameters may be added to ISUP messages by the MTP.

b.1.2. Local Management Primitives

b.1.2.1. MTP_INFO_ACK

Parameters

Flags

Rules

b.1.2.2. MTP_BIND_REQ

Parameters

mtp_addr_length:	Indicates the length of the address to bind.
mtp_addr_offset:	Indicates the offset of the address to bind from the beginning of the block.
mtp_setup_ind:	Indicates the maximum number of setup (or continuity check) indications that will be oustanding for the listening stream.

Flags

MTP_DEFAULT_LISTENER	Specifies that the stream to be bound is the MTP_DEFAULT_LISTENER stream.
MTP_CIRCUIT	Specifies that the address to be bound is to be interpreted by the MTP provider as a circuit identifier.
MTP_GROUP	Specifies that the address to be bound is a to be interpreted by the MTP provider as a trunk group identifier.

Rules

Rules for address specification:

(1) The address contained in the primitive as indicated by *mtp_addr_length* and *mtp_addr_offset* parameters must be either a trunk group identifier or a circuit identifier (as indicated by the flags MTP_GROUP or MTP_CIRCUIT).

(2) If the MTP_DEFAULT_LISTENER flag is set, the parameters *mtp_addr_length* and *mtp_addr_offset* should be coded zero, and should be ignored by the MTP provider.

Rules for setup indications:

(1) If the number of setup indications is non-zero, the stream is bound as a listening stream. Listening streams will receive all calls, test calls, and continuity tests that are incoming on the address bound.

- If the address bound is a circuit (MTP_CIRCUIT flag set), all such incoming calls on thecircuit will be delivered to the stream listening on the circuit.
- If the address bound is a trunk group (MTP_GROUP flag set), all such incoming calls on any circuit of the trunk group (which have no other stream bound as a listener to the circuit) will be delivered to stream listening on the trunk group.
- Streams that are listening bound to trunk groups (MTP_GROUP flag set) will have a maximum number of setup indicaitons no more that the number of circuits in the trunk group.
- Streams that are listening bound to a circuit (MTP_CIRCUIT flag set) will have a maximum number of setup indications of one (1).

(2) Once a stream has successfully bound as a listening stream, it should be prepared to receive incoming calls, test calls and continuity tests.

Rules for bind flags:

(1) For Q.764 conformance, the MTP_DEFAULT_LISTENER will receieve all incoming calls, test calls and continuity tests that have no other listening stream. There can only be one stream bound with the MTP_DEFAULT_LISTENER flag set.

(2) Only one of MTP_DEFAULT_LISTENER, MTP_CIRCUIT and MTP_GROUP may be set.

b.1.2.3. MTP_BIND_ACK

Parameters

mtp_addr_length:	Indicates the length of the address to bind.
mtp_addr_offset:	Indicates the offset of the address to bind from the beginning of the block.
mtp_setup_ind:	Indicates the maximum number of setup (or continuity check) indications that will be outstanding for the listening stream.

Flags

MTP_DEFAULT_LISTENER	Specifies that the stream to be bound is the MTP_DEFAULT_LISTENER stream.
MTP_CIRCUIT	Specifies that the address to be bound is to be interpreted by the MTP provider as a circuit identifier.
MTP_GROUP	Specifies that the address to be bound is a to be interpreted by the MTP provider as a trunk group identifier.

Rules

Rules for address specification:

(1) The address contained in the primitive as indicated by *mtp_addr_length* and *mtp_addr_offset* parameters must be either a trunk group identifier or a circuit identifier (as indicated by the flags MTP_GROUP or MTP_CIRCUIT).

(2) If the MTP_DEFAULT_LISTENER flag is set, the parameters *mtp_addr_length* and *mtp_addr_offset* will be coded zero, and should be ignored by the MTP-User.

Rules for setup indications:

(1) If the number of setup indications is non-zero, the stream is bound as a listening stream. Listening streams will receive all calls, test calls, and continuity tests that are incoming on the address bound.

- If the address bound is a circuit (MTP_CIRCUIT flag set), all such incoming calls on thecircuit will be delivered to the stream listening on the circuit.
- If the address bound is a trunk group (MTP_GROUP flag set), all such incoming calls on any circuit of the trunk group (which have no other stream bound as a listener to the circuit) will be delivered to stream listening on the trunk group.
- Streams that are listening bound to trunk groups (MTP_GROUP flag set) will have a maximum number of setup indicaitons no more that the number of circuits in the trunk group.

  - Streams that are listening bound to a circuit (MTP_CIRCUIT flag set) will have a maximum number of setup indications of one (1).

(2) Once a stream has successfully bound as a listening stream, the provider should be prepared to send all incoming calls, test calls and continuity tests to the bound stream.

Rules for bind flags:

(1) For Q.764 conformance, the MTP_DEFAULT_LISTENER will receieve all incoming calls, test calls and continuity tests that have no other listening stream. There can only be one stream bound with the MTP_DEFAULT_LISTENER flag set.

(2) Only one of MTP_DEFAULT_LISTENER, MTP_CIRCUIT and MTP_GROUP will be set.

b.1.2.4. MTP_OPTMGMT_REQ

Parameters

Flags

Rules

b.1.3. Call Setup Primitives

b.1.3.1. MTP_SETUP_REQ

Parameters

mtp_call_type: Specifies the type of call to be set up. For Q.764 conforming MTP providers, the call type can be one of the following values:

MTP_CALL_TYPE_SPEECH
: The call type is speech. This call type corresponds to a Q.764 transmission medium requirement of Speech.

MTP_CALL_TYPE_64KBS_UNRESTRICTED
: The call type is 64 kbit/s unrestricted digital information. This call type corresponds to a Q.764 transmission medium requirement of 64 kbit/s Unrestricted Digital Information.

MTP_CALL_TYPE_3_1kHZ_AUDIO
: The call type is 3.1 kHz audio. This call type corresponds to a Q.764 transmission medium requirement of 3.1 kHz Audio.

MTP_CALL_TYPE_64KBS_PREFERRED
: The call type is 64 kbit/s preferred. This call type corresponds to a Q.764 transmission medium requirement of 64 kbit/s Preferred.

MTP_CALL_TYPE_2x64KBS_UNRESTRICTED
: The call type is 2 x 64 kbit/s unrestricted digital information. This call type corresponds to a Q.764 transmission medium requirement of 2 x 64 kbit/s Unrestricted Digital Information.

MTP_CALL_TYPE_384KBS_UNRESTRICTED
: The call type is 384 kbit/s unrestricted digital information. This call type corresponds to a Q.764 transmission medium requirement of 384 kbit/s Unrestricted Digital Information.

MTP_CALL_TYPE_1536KBS_UNRESTRICTED
: The call type is 1536 kbit/s unrestricted digital information. This call type corresponds to a Q.764 transmission medium requirement of 1536 kbit/s Unrestricted Digital Information.

MTP_CALL_TYPE_1920KBS_UNRESTRICTED
: The call type is 1920 kbit/s unrestricted digital information. This call type corresponds to a Q.764 transmission medium requirement of 1920 kbit/s Unrestricted Digital Information.

mtp_call_flags:	Specifies the options associated with the call. (See "Flags" below.)
mtp_cdpn_length:	Specifies the length of the called party number. For Q.764 conforming MTP providers, the format of the called party number is the format of the Called Party Number parameter (without the parameter type or length octets) as specified in Q.763.
mtp_cdpn_offset:	Specifies the offset of the called party number from the beginning of the block.

Flags

Q.764 conforming MTP providers must support the following flags:

The following flags correspond to bits in the Nature of Connection Indicators parameter of Q.763:

MTP_NCI_ONE_SATELLITE_CCT

MTP_NCI_TWO_SATELLITE_CCT
When one of these flags is set it indicates that either one or two satellite circuits are present in the connection. Otherwise, it indicates that no satellite circuits are present in the connection.

MTP_NCI_CONT_CHECK_REQUIRED

MTP_NCI_CONT_CHECK_PREVIOUS
When one of these flags is set it indicates that either a continuity check is required on the connection, or that a continuity check was performed on a previous connection. Otherwise, it indicates that a continuity check is not required on the connection.

MTP_NCI_OG_ECHO_CONTROL_DEVICE
When set it indicates that an outgoing half echo control device is included on the connection. Otherwise, it indicates that no outgoing half echo control device is included on the connection.

The following flags correspond to bits in the Forward Call Indicators parameter of Q.763:

MTP_FCI_INTERNATIONAL_CALL
When this flag is set, the call is to be treated as an international call. Otherwise, the call is to be treated as a national call.

MTP_FCI_PASS_ALONG_E2E_METHOD_AVAILABLE

MTP_FCI_SCCP_E2E_METHOD_AVAILABLE
When one of these flags is set, either the pass along end-to-end method is available or the SCCP end-to-end method is available. Otherwise, no end-to-end method is available and only link-by-link method is available.

MTP_FCI_INTERWORKING_ENCOUNTERED
When this flag is set, interworking has been encountered on the call. Otherwise, no interworking has been encountered on the call.

MTP_FCI_E2E_INFORMATION_AVAILABLE
When this flag is set, end-to-end information is now available. Otherwise, no end-to-end information is available.

MTP_FCI_ISDN_USER_PART_ALL_THE_WAY
When this flag is set, ISDN User Part has been used all the way on the call. Otherwise, ISDN User Part has not been used all the way.

MTP_FCI_ORIGINATING_ACCESS_ISDN
When this flag is set, the originating access is ISDN. Otherwise, the originating access is non-ISDN.

MTP_FCI_SCCP_CLNS_METHOD_AVAILABLE

MTP_FCI_SCCP_CONS_METHOD_AVAILABLE

MTP_FCI_SCCP_ALL_METHODS_AVAILABLE
When one of these flags is set, either the connectionless SCCP method is available, the connection oriented SCCP method is available, or both methods are available. Otherwise, no SCCP method is indicated as available.

Rules

Rules for call type:

(1) All Q.764 conforming MTP must support the following call types:

MTP_CALL_TYPE_SPEECH,
MTP_CALL_TYPE_64KBS_UNRESTRICTED,
MTP_CALL_TYPE_3_1kHZ_AUDIO, and
MTP_CALL_TYPE_64KBS_PREFERRED.

(2) Support for other call types is optional and implementation-specific.

Rules for flags:

(1) Q.764 conforming MTP providers must support all of the flags listed above.

(2) Only one of the following flags may be set:

MTP_NCI_ONE_SATELLITE and
MTP_NCI_TWO_SATELLITE.

(3) Only one of the following flags may be set:

MTP_NCI_CONT_CHECK_REQUIRED and
MTP_NCI_CONT_CHECK_PREVIOUS.

(4) Only one of the following flags may be set:

MTP_FCI_PASS_ALONG_E2E_METHOD_AVAILABLE and
MTP_FCI_SCCP_E2E_METHOD_AVAILABLE.

(5) Only one of the following flags may be set, and only if MTP_CFI_SCCP_E2E_METHOD_AVAILABLE is also set:

MTP_FCI_SCCP_CLNS_METHOD_AVAILABLE,
MTP_FCI_SCCP_CONS_METHOD_AVAILABLE and
MTP_FCI_SCCP_ALL_METHODS_AVAILABLE.

b.1.3.2. MTP_SETUP_IND

Parameters

mtp_call_type	Indicates the type of call to be set up. For Q.764 conforming MTP providers, the call type can be one of the call types listed in this addendum under MTP_SETUP_REQ.
mtp_call_flags	Indicates the options associated with the call. (See "Flags" below.)
mtp_addr_length	Indicates the length of the identifier of the circuit upon which the call setup is indicated.
mtp_addr_offset	Indicates the offset of the identifier from the start of the block.
mtp_cdpn_length	Indicates the length of the called party number. For Q.764 conforming MTP providers, the format of the called party number is the format of the Called Party Number parameter (without the parameter type or length octets) as specified in Q.763.
mtp_cdpn_offset	Indicates the offset of the called party number from the beginning of the block.

Flags

Q.764 conforming MTP providers indicate the flags listed in this addendum under MTP_SETUP_REQ.

Rules

Rules for call type:

(1) The rules for call type in section MTP_SETUP_REQ in this addendum also apply to the MTP_SETUP_IND.

Rules for setup flags:

(1) The rules for setup flags in section MTP_SETUP_REQ in this addendum also apply to the MTP_SETUP_IND.

b.1.3.3. MTP_SETUP_RES

Parameters

mtp_event:	Indicates the event and associated optional paramters upon which the call was accepted.

Rules

b.1.3.4. MTP_SETUP_CON

Parameters

mtp_addr_length:	Indicates the length of the identifier of the circuit upon which the call setup is indicated.
mtp_addr_offset:	Indicates the offset of the identifier from the start of the block.

Rules

Rules for addresses:

(1) The address indicated in the MTP_SETUP_CON is the circuit identifier of the outgoing circuit upon which the call has been connected.

b.1.3.5. MTP_CALL_REATTEMPT_IND

Parameters

mtp_cause Indicates the cause of the reattempt. The mtp_cause can have one of the following values:

MTP_REATTEMPT_DUAL_SEIZURE
: Indicates that the circuit was seized by a controlling exchange during the initial setup of the call (i.e, before any backward message was received).

MTP_REATTEMPT_RESET
: Indicates that the circuit was reset during the initial setup of the call (i.e, before any backward message was received).

MTP_REATTEMPT_BLOCKING
: Indicates that the circuit was blocked during the initial setup of the call (i.e, before any backward message was received).

MTP_REATTEMPT_T24_TIMEOUT
: Indicates that COT failure occured on the circuit.

MTP_REATTEMPT_UNEXPECTED
: Indicates that an unexpected message was received for the call during the initial setup of the call (i.e, before any backward message was received).

MTP_REATTEMPT_COT_FAILURE
: Indicates that COT failed on the circuit.

mtp_addr_length Indicates the length of the identifier of the circuit upon which the reattempt will be performed.

mtp_addr_offset Indicates the offset of the address from the start of the block.

Rules

b.1.3.6. MTP_SETUP_COMPLETE_REQ

Rules

For compatibility betweem MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if an MTP conforming to Q.764 receives a MTP_SETUP_COMPLETE_REQ in the MTPS_ANSWERED state (MTPS_ICC_ANSWERED), the MTP provider should ignore the primitive.

b.1.4. Continuity Check Phase

b.1.4.1. MTP_CONT_CHECK_REQ

Parameters

mtp_addr_length:	Specifies the length of the identifier of the circuit upon which the continuity check is to be performed.
mtp_addr_offset:	Specifies the offset of the address from the start of the block.

Rules

Rules for addresses:

(1) The parameter *mtp_addr_length* cannot be zero: i.e, an address must be provided or the MTP should respond with MTP_ERROR_ACK with an error of MNOADDR.

(2) The address provided must be the identifier of the circuit upon which the MTP-User is requesting a continuity check.

(3) The specified circuit identifier must be equipped else the MTP should response with MTP_ERROR_ACK and an error of MBADADDR.

b.1.4.2. MTP_CONT_CHECK_IND

Parameters

mtp_addr_length:	Specifies the length of the identifier of the circuit upon which the continuity check is to be performed.
mtp_addr_offset:	Specifies the offset of the address from the start of the block.

Rules

Rules for addresses:

(1) The parameter *mtp_addr_length* cannot be zero: i.e, an address must be provided or the MTP should respond with MTP_ERROR_ACK with an error of MNOADDR.

(2) The address provided must be the identifier of the circuit upon which the MTP-User is requesting a continuity check.

(3) The specified circuit identifier must be equipped else the MTP should response with MTP_ERROR_ACK and an error of MBADADDR.

b.1.4.3. MTP_CONT_CHECK_RES

This primitive is only supported when the Loopback Acknowledgement is used as a national option under Q.764. For compatibility with MTP providers not supporting the national option, if such an MTP receives a MTP_CONT_CHECK_RES while waiting for an MTP_CONT_REPORT_IND, the MTP should silently discard the primitive.

Parameters

mtp_addr_length:	Specifies the length of the identifier of the circuit upon which the continuity check is to be performed.

Rules

Rules for addresses:

(1) The parameter *mtp_addr_length* cannot be zero: i.e, an address must be provided or the MTP should respond with MTP_ERROR_ACK with an error of MNOADDR.

(2) The address provided must be the identifier of the circuit upon which the MTP-User is requesting a continuity check.

(3) The specified circuit identifier must be equipped else the MTP should response with MTP_ERROR_ACK and an error of MBADADDR.

b.1.4.4. MTP_CONT_CHECK_CON

This primitive is only supported when the Loopback Acknowledgement is used as a national option under Q.764. For compatibility with MTP providers not supporting the national option, such an MTP will issue a MTP_CONT_CHECK_CON in response to an MTP_CONT_CHECK_REQ following the MTP_OK_ACK.

Parameters

mtp_addr_length:	Specifies the length of the identifier of the circuit upon which the continuity check is to be performed.
mtp_addr_offset:	Specifies the offset of the address from the start of the block.

Rules

Rules for addresses:

(1) The parameter *mtp_addr_length* cannot be zero: i.e, an address must be provided or the MTP should respond with MTP_ERROR_ACK with an error of MNOADDR.

(2) The address provided must be the identifier of the circuit upon which the MTP-User is requesting a continuity check.

(3) The specified circuit identifier must be equipped else the MTP should response with MTP_ERROR_ACK and an error of MBADADDR.

b.1.4.5. MTP_CONT_REPORT_REQ

Parameters

mtp_result: Indicates the result of the continuity test, whether success or failure. For Q.764 conforming MTP, the result paramter can be one of the following values:

MTP_COT_SUCCESS
Indicates that the continuity check test was successful.

MTP_COT_FAILURE
Indicates that the continuity check test failed.

mtp_addr_length: Specifies the length of the identifier of the circuit upon which the continuity check is to be performed.

mtp_addr_offset: Specifies the offset of the address from the start of the block.

Rules

Rules for addresses:

(1) The parameter *mtp_addr_length* cannot be zero: i.e, an address must be provided or the MTP should respond with MTP_ERROR_ACK with an error of MNOADDR.

(2) The address provided must be the identifier of the circuit upon which the MTP-User is requesting a continuity check.

(3) The specified circuit identifier must be equipped else the MTP should response with MTP_ERROR_ACK and an error of MBADADDR.

b.1.4.6. MTP_CONT_REPORT_IND

Parameters

mtp_result: Indicates the result of the continuity test, whether success or failure. For Q.764 conforming MTP, the result paramter can be one of the following values:

MTP_COT_SUCCESS
Indicates that the continuity check test was successful.

MTP_COT_FAILURE
Indicates that the continuity check test failed.

mtp_addr_length: Specifies the length of the identifier of the circuit upon which the continuity check is to be performed.

mtp_addr_offset: Specifies the offset of the address from the start of the block.

Rules

Rules for addresses:

(1) The parameter *mtp_addr_length* cannot be zero: i.e, an address must be provided or the MTP should respond with MTP_ERROR_ACK with an error of MNOADDR.

(2) The address provided must be the identifier of the circuit upon which the MTP-User is requesting a continuity check.

(3) The specified circuit identifier must be equipped else the MTP should response with MTP_ERROR_ACK and an error of MBADADDR.

b.1.5. Call Establishment Primitives

b.1.5.1. MTP_MORE_INFO_REQ

Rules

Rules for issuing primitive:

This primitive is not directly supported by Q.764 conforming MTP providers. For compatibility with Q.931 conforming MTP providers, if the Q.764 conforming MTP receives an MTP_MORE_INFO_REQ in state MTPS_WRES_SIND, it should invoke any interworking procedures and silently discard the primitive.

b.1.5.2. MTP_MORE_INFO_IND

Rules

Rules for issuing primitive:

(1) This primitive may optionally be issued by a Q.764 conforming MTP in the *overlap* signalling mode, if the appropriate timer has expired and the MTP has not received an indication that the provided address is complete.

b.1.5.3. MTP_INFORMATION_REQ

Parameters

mtp_subn_length:	Indicates the length of the subsequent number. For Q.764 conforming MTP providers, the format of the called party address is the format of the Subsequent Number parameter (without the parameter type or length octets) as specified in Q.763.
mtp_subn_offset:	Indicates the offset of the subsequent number from the beginning of the block.

Rules

Rules for issuing primitive:

(1) This primitive will only be issued before any MTP_PROCEEDING_IND, MTP_ALERTING_IND, MTP_PROGRESS_IND, or MTP_IBI_IND has occured on the stream while in the MTPS_WCON_SREQ state. If not, the MTP should respond with an MTP_ERROR_ACK primitive with error MOUTSTATE.

(2) This primitive must not be issued if the preceeding MTP_SETUP_REQ contained a called party address which was complete (i.e, contains a ST code following the digits). If it is, the MTP should response with an MTP_ERROR_ACK with error MBADADDR.

(3) This primitive must not be issued if the trunk group or circuit to which the stream is bound is configured for *en bloc* operation. If it is, the MTP should response with an MTP_ERROR_ACK with error MNOTSUPPORT.

b.1.5.4. MTP_INFORMATION_IND

Parameters

mtp_subn_length:	Indicates the length of the subsequent number. For Q.764 conforming MTP providers, the format of the subsequent number is the format of the Subsequent Number parameter (without the parameter type or length octets) as specified in Q.763.
mtp_subn_offset:	Indicates the offset of the subsequent number from the beginning of the block.

Rules

Rules for issuing primitive:

(1) This primitive will only be issued by the MTP before any MTP_PROCEEDING_REQ, MTP_ALERTING_REQ, MTP_PROGRESS_REQ, or MTP_IBI_REQ has been received in state MTPS_WCON_SREQ.

(2) This primitive will not be issued by the MTP if the preceeding MTP_SETUP_REQ contained a complete called party address (i.e, contains an ST code following the digits), or if the trunk group or circuit is configured for *en bloc* operation.

b.1.5.5. MTP_INFO_TIMEOUT_IND

Rules

Rules for issuing primitive:

(1) If the Q.764 conforming MTP encounters interworking on a call and is not expecting an address complete message, and timer T11 expires, the MTP provider will issue this primitive to the MTP-User.

(2) Upon receipt of this primitive, it is the MTP-User's responsibility to determine whether the address digits are sufficient and to issue a MTP_SETUP_RES or MTP_REJECT_REQ primitive.

For compatibility between MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if the MTP-User receives an MTP_INFO_TIMEOUT_IND

b.1.5.6. MTP_PROCEEDING_REQ

Parameters

mtp_flags: Specifies the options associated with the call. (See "Flags" below.) Indicatest the flags associated with the primitive. For Q.764 conforming MTP providers, call flags can be an of the following:

Flags

Q.764 conforming MTP must suppor the following flags:

The following flags correspond to bits in the Backward Call Indicators parameter of Q.763:

MTP_BCI_NO_CHARGE

MTP_BCI_CHARGE
: When one of these flags is set, it indicates that the call is not to be charged, or the call is to be charged. Otherwise, it indicates that there is no indication with regard to charging.

MTP_BCI_SUBSCRIBER_FREE

MTP_BCI_CONNECT_FREE
: When one of these flags is set, it indicates that the terminating subscriber is free, or that the connection is free. Otherwise, no indication is given.

MTP_BCI_ORDINARY_SUBSCRIBER

MTP_BCI_PAYPHONE
: When one of these flags is set, it indicates that the call has terminated to an ordinary subscriber, or that the call has terminated to a payphone.

MTP_BCI_PASS_ALONG_E2E_METHOD_AVAILABLE

MTP_BCI_SCCP_E2E_METHOD_AVAILABLE
: When one of these flags is set, either the pass along end-to-end method is available, or the SCCP end-to-end method is available. Otherwise, no end-to-end method is available and only link-by-link method is available.

MTP_BCI_INTERWORKING_ENCOUNTERED
: When this flag is set, interworking has been encountered on the call. Otherwise, to interworking has been encountered on the call.

MTP_BCI_E2E_INFORMATION_AVAILABLE
: When this flag is set, end-to-end information is now available. Otherwise, no end-to-end information is available.

MTP_BCI_ISDN_USER_PART_ALL_THE_WAY
: When this flag is set, ISDN User Part has been used all the way on the call, Otherwise, ISDN User Part has not be used all the way.

MTP_BCI_HOLDING_REQUESTED
: When this flag is set, hodling is requested. Otherwise, holding is not requested.

MTP_BCI_TERMINATING_ACCESS_ISDN
: When this flag is set, the terminating access is ISDN. Otherwise, the terminating access is non-ISDN.

MTP_BCI_IC_ECHO_CONTROL_DEVICE
: When set, this flag indicates that an incoming half echo control device is included on the connection. Otherwise, it indicates that no incoming half echo control device is included in the connection.

MTP_BCI_SCCP_CLNS_METHOD_AVAILABLE

MTP_BCI_SCCP_CONS_METHOD_AVAILABLE

MTP_BCI_SCCP_ALL_METHODS_AVAILABLE
: When one of these flags is set, either the connectionless SCCP method is available, the connection oriented SCCP method is available, or both methods are available. Otherwise, no SCCP method is indicated as available.

Rules

Rules for issuing primitive:

(1) This primitive can only be issued by the MTP-User before any MTP_ALERTING_REQ, MTP_PROGRESS_REQ or MTP_IBI_REQ has been issued while in state MTPS_WRES_SIND.

b.1.5.7. MTP_PROCEEDING_IND

Rules

Rules for issuing primitive:

(1) This primitive will only be issued by the MTP before any MTP_ALERTING_IND, MTP_PROGRESS_IND or MTP_IBI_IND has been issued while in state MTPS_WCON_SREQ.

b.1.5.8. MTP_ALERTING_REQ

Rules

Rules for issuing primitive:

(1) This primitive can only be issued by the MTP-User before any MTP_PROGRESS_REQ or MTP_IBI_REQ has been issued while in state MTPS_WRES_SIND.

b.1.5.9. MTP_ALERTING_IND

Rules

Rules for issuing primitive:

(1) This primitive will only be issued by the MTP before any MTP_PROGRESS_IND or MTP_IBI_IND has been issued while in state MTPS_WCON_SREQ.

b.1.5.10. MTP_PROGRESS_REQ

Parameters

mtp_event: Indicates the progress event. For Q.764 conforming MTP providers, this can be one of the following:

MTP_EVENT_ALERTING
: Indicates that the called party is being alerted. This event is indicated only if an MTP_CALL_PROCEEDING_IND primitive has already been received.

MTP_EVENT_PROGRESS
: Indicates that the call is progressing with the specified optional parameters.

MTP_EVENT_IBI
: This event is indicated only by the MTP_IBI_IND primitive and will not appear here.

MTP_EVENT_CALL_FORWARDED_ON_BUSY
: This event indicates that the call has been forwarded on busy and the optional parameters (if any) contain the attributes of the forwarding (e.g., redirecting number, etc.).

MTP_EVENT_CALL_FORWARDED_ON_NO_ANSWER
: This event indicates that the call has been forwarded on no answer and the optional parameters (if any) contain the attributes of the forwarding (e.g., redirecting number, etc.).

MTP_EVENT_CALL_FORWARDED_UNCONDITIONAL
: This event indicates that the call has been forwarded unconditionally and the optional parameters (if any) contain the attributes of the forwarding (e.g., redirecting number, etc.).

mtp_flags: Indicates the options flags. (See "Flags" below.)

Flags

MTP_EVENT_PRESENTATION_RESTRICTED
: When set, this flag indicates that the event indication is not to be presented to the caller. Otherwise, the event may be presented to the caller.

Rules

Rules for issuing primitive:

(1) This primitive can only be issued by the MTP-User before any MTP_IBI_REQ has been issued while in state MTPS_WRES_SIND.

Rules for progress event:

(1) Q.764 conforming MTP providers must support the complete list of progress events listed above.

(2) When this primitive is issued with the event MTP_EVENT_ALERTING, it must follow the rules for the primitive MTP_ALERTING_REQ.

(3) When this primitive is issued with the event MTP_EVENT_IBI, it must follow the rules for the primitive MTP_IBI_REQ.

Rules for progress flags:

(1) The flag MTP_EVENT_PRESENTATION_RESTRICTED cannot be set when the event is MTP_EVENT_ALERTING, MTP_EVENT_PROGRESS or MTP_EVENT_IBI.

b.1.5.11. MTP_PROGRESS_IND

Parameters

mtp_event: Indicates the progress event. The event can be any of the events listed in this addendum under MTP_PROGRES_REQ.

mtp_flags: Indicates the options flags. (See "Flags" below.)

Flags

MTP_EVENT_PRESENTATION_RESTRICTED
: When set, this flag indicates that the event indication is not to be presented to the caller. Otherwise, the event may be presented to the caller.

Rules

Rules for issuing primitive:

(1) This primitive will only be issued by the MTP before any MTP_IBI_IND has been issued while in state MTPS_WCON_SREQ.

Rules for progress event:

(1) Q.764 conforming MTP providers must support the complete list of progress events listed above.

(2) This primitive will not be issued by the MTP with event MTP_EVENT_ALERTING or event MTP_EVENT_IBI: instead, an MTP_ALERTING_IND or MTP_IBI_IND event will be issued.

Rules for progress flags:

(1) The flag MTP_EVENT_PRESENTATION_RESTRICTED cannot be set when the vent is MTP_EVENT_PROGRESS.

b.1.5.12. MTP_IBI_REQ

Rules

b.1.5.13. MTP_IBI_IND

Rules

b.1.6. Call Established Primitives

b.1.6.1. MTP_SUSPEND_REQ

Parameters

mtp_flags:	Specifies options associated with the suspend. See "Flags" below.

Flags

MTP_SUSRES_NETWORK_INITIATED
: When this flag is set, it indicates that the suspend was network originated. When this flag is not set, it indicates that the suspend was ISDN subscriber initiated.

Rules

Rules for issuing primitive:

(1) For Q.764 conforming MTP providers, suspend can be requested by independently either via local provider or the remote provider. A call can be:

- Not Suspended
- Locally Suspended
- Remotely Suspended
- Locally and Remotely Suspended

(2) Requests to locally suspend a call which is already locally suspended should be ignored by the MTP.

b.1.6.2. MTP_SUSPEND_IND

Parameters

mtp_flags:	Specifies options associated with the suspend. See "Flags" below.

Flags

MTP_SUSRES_NETWORK_INITIATED
: When this flag is set, it indicates that the suspend was network originated. When this flag is not set, it indicates that the suspend was ISDN subscriber initiated.

Rules

Rules for issuing primitive:

(1) For Q.764 conforming MTP providers, suspend can be requested by independently either via local provider or the remote provider. A call can be:

- Not Suspended
- Locally Suspended
- Remotely Suspended
- Locally and Remotely Suspended

(2) Indications of remote suspension of a call which is already remotely suspended will not be issued by the MTP.

b.1.6.3. MTP_SUSPEND_RES

Rules

Rules for issuing primitive:

For compatibility between MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if the MTP receives an MTP_SUSPEND_RES in the MTPS_WRES_SUSIND or MTPS_SUSPENDED states, the MTP should ignore the MTP_SUSPEND_RES primitive and move directly to the MTPS_SUSPENDED state if it has not already done so.

b.1.6.4. MTP_SUSPEND_REJECT_REQ

Rules

Rules for issuing primitive:

For compatibility between MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if the MTP receives an MTP_SUSPEND_REJECT_REQ in the MTPS_WRES_SUSIND or MTPS_SUSPENDED states, the MTP should reply with an MTP_ERROR_ACK primitive with error MNOTSUPP.

b.1.6.5. MTP_RESUME_REQ

Parameters

mtp_flags:	Specifies options associated with the resume. See "Flags" below.

Flags

MTP_SUSRES_NETWORK_INITIATED
When this flag is set, it indicates that the resume was network originated. When this flag is not set, it indicates that the resume was ISDN subscriber initiated.

Rules

b.1.6.6. MTP_RESUME_IND

Parameters

mtp_flags:	Specifies options associated with the resume. See "Flags" below.

Flags

MTP_SUSRES_NETWORK_INITIATED
When this flag is set, it indicates that the resume was network originated. When this flag is not set, it indicates that the resume was ISDN subscriber initiated.

Rules

b.1.6.7. MTP_RESUME_RES

Rules

Rules for issuing primitive:

For compatibility between MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if the MTP receives an MTP_RESUME_RES in the MTPS_WRES_SUSIND or MTPS_ANSWERED states, the MTP should ignore the MTP_RESUME_RES primitive and move directly to the MTPS_RESUMEED state if it has not already done so.

b.1.6.8. MTP_RESUME_REJECT_REQ

Rules

Rules for issuing primitive:

For compatibility between MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if the MTP receives an MTP_RESUME_REJECT_REQ in the MTPS_WRES_SUSIND or MTPS_ANSWERED states, the MTP should reply with an MTP_ERROR_ACK primitive with error MNOTSUPP.

b.1.7. Call Termination Primitives

b.1.7.1. MTP_REJECT_REQ

Rules

Rules for issuing primitive:

For compatibility between MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if the MTP receives an MTP_REJECT_REQ in the MTPS_WRES_SIND (MTPS_ICC_WAIT_COT or MTPS_ICC_WAIT_ACM) states, the provider should perform an automatic release procedure and move to the MTPS_WAIT_RLC state.

b.1.7.2. MTP_CALL_FAILURE_IND

Parameters

mtp_cause: Indicates the cause of the failure. The mtp_cause can have one of the following values:

MTP_FAILURE_COT_FAILURE
: Indicates that the continuity check on the circuit failed. This applies to incoming calls only.

MTP_FAILURE_RECV_RLC
: Indicates that the circuit was not completely released by the distant end. This applies to incoming calls only.

MTP_FAILURE_BLOCKING
: Indicates that the circuit was blocked during call setup. This applies to incoming calls only.

MTP_FAILURE_T6_TIMEOUT
: Indicates that the call was suspended beyond the allowable period. This applies to all established calls.

MTP_FAILURE_T7_TIMEOUT
: Indicates that there was no response to the call setup request. This applies to outgoing calls only.

MTP_FAILURE_T8_TIMEOUT
: Indicates that the call failed waiting for a continuity check report from the distant end. This applies to incoming calls only.

MTP_FAILURE_T9_TIMEOUT
: Indicates that the call failed while waiting for the distant end to answer. This applies to outgoing calls only.

MTP_FAILURE_T35_TIMEOUT
: Indicates that additional information (digits) were not received from the caller within a sufficient period. This applies to incoming calls only.

MTP_FAILURE_T38_TIMEOUT
: Indicates that the call was suspended beyond the allowble period. This applies to all established calls.

Rules

b.1.7.3. MTP_DISCONNECT_REQ

Rules

For compatability between MTP providers conforming to Q.931 and MTP providers conforming to Q.764, if the MTP receives an MTP_DISCONNECT_REQ, the provider should respond with MTP_ERROR_ACK with the error MNOTSUPPORT.

b.1.7.4. MTP_RELEASE_REQ

Parameters

mtp_cause:	Indicates the cause of the release. Cause can be one of the following values:
	MTP_CAUS_UNALLOCATED_NUMBER
	MTP_CAUS_NO_ROUTE_TO_TRANSIT_NETWORK
	MTP_CAUS_NO_ROUTE_TO_DESTINATION
	MTP_CAUS_SEND_SPECIAL_INFO_TONE
	MTP_CAUS_MISDIALLED_TRUNK_PREFIX
	MTP_CAUS_PREEMPTION
	MTP_CAUS_PREEMPTION_CCT_RESERVED
	MTP_CAUS_NORMAL_CALL_CLEARING
	MTP_CAUS_USER_BUSY
	MTP_CAUS_NO_USER_RESPONDING
	MTP_CAUS_NO_ANSWER
	MTP_CAUS_SUBSCRIBER_ABSENT
	MTP_CAUS_CALL_REJECTED
	MTP_CAUS_NUMBER_CHANGED
	MTP_CAUS_REDIRECT
	MTP_CAUS_OUT_OF_ORDER
	MTP_CAUS_ADDRESS_INCOMPLETE
	MTP_CAUS_FACILITY_REJECTED
	MTP_CAUS_NORMAL_UNSPECIFIED
	MTP_CAUS_NO_CCT_AVAILABLE
	MTP_CAUS_NETWORK_OUT_OF_ORDER
	MTP_CAUS_TEMPORARY_FAILURE
	MTP_CAUS_SWITCHING_EQUIP_CONGESTION
	MTP_CAUS_ACCESS_INFO_DISCARDED
	MTP_CAUS_REQUESTED_CCT_UNAVAILABLE
	MTP_CAUS_PRECEDENCE_CALL_BLOCKED
	MTP_CAUS_RESOURCE_UNAVAILABLE
	MTP_CAUS_NOT_SUBSCRIBED
	MTP_CAUS_OGC_BARRED_WITHIN_CUG
	MTP_CAUS_ICC_BARRED WITHIN_CUG
	MTP_CAUS_BC_NOT_AUTHORIZED
	MTP_CAUS_BC_NOT_AVAILABLE
	MTP_CAUS_INCONSISTENCY
	MTP_CAUS_SERVICE_OPTION_NOT_AVAILABLE
	MTP_CAUS_BC_NOT_IMPLEMENTED
	MTP_CAUS_FACILITY_NOT_IMPLEMENTED
	MTP_CAUS_RESTRICTED_BC_ONLY
	MTP_CAUS_SERIVCE_OPTION_NOT_IMPLEMENTED
	MTP_CAUS_USER_NOT_MEMBER_OF_CUG

MTP_CAUS_INCOMPATIBLE_DESTINATION
MTP_CAUS_NON_EXISTENT_CUG
MTP_CAUS_INVALID_TRANSIT_NTWK_SELECTION
MTP_CAUS_INVALID_MESSAGE
MTP_CAUS_MESSAGE_TYPE_NOT_IMPLEMENTED
MTP_CAUS_PARAMETER_NOT_IMPLEMENTED
MTP_CAUS_RECOVERY_ON_TIMER_EXPIRY
MTP_CAUS_PARAMETER_PASSED_ON
MTP_CAUS_MESSAGE_DISCARDED
MTP_CAUS_PROTOCOL_ERROR
MTP_CAUS_INTERWORKING
MTP_CAUS_UNALLOCATED_DEST_NUMBER
MTP_CAUS_UNKNOWN_BUSINESS_GROUP
MTP_CAUS_EXCHANGE_ROUTING_ERROR
MTP_CAUS_MISROUTED_CALL_TO_PORTED_NUMBER 26
MTP_CAUS_LNP_QOR_NUMBER_NOT_FOUND
MTP_CAUS_PREEMPTION
MTP_CAUS_PRECEDENCE_CALL_BLOCKED
MTP_CAUS_CALL_TYPE_INCOMPATIBLE
MTP_CAUS_GROUP_RESTRICTIONS

Rules

b.1.7.5. MTP_RELEASE_IND

Parameters

mtp_cause:	Indicates the cause of the release. Cause can be one of the cause valuse listed in this addendum under MTP_RELEASE_REQ.

Rules

b.1.8. Management Primitives

b.1.8.1. MTP_RESTART_REQ

Rules

For compatability between MTP providers conforming to Q.931 and MTP conforming to Q.764, if the MTP conforming to Q.764 receives a MTP_RESTART_REQ, the provider should respond with MTP_ERROR_ACK with the error MNOTSUPPORT.

b.1.8.2. MTP_RESET_REQ

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

Rules

b.1.8.3. MTP_RESET_IND

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

Rules

b.1.8.4. MTP_RESET_RES

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

Rules

b.1.8.5. MTP_RESET_CON

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

Rules

b.1.8.6. MTP_BLOCKING_REQ

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

MTP_MAINTENANCE_ORIENTED

MTP_HARDWARE_FAILURE_ORIENTED
: When one of these flags is set it indicates that either maintenance oriented or hardware failure oriented blocking is to be performed. If both or neither of these flags are set, the primitive will fail with error MBADFLAG.

Rules

b.1.8.7. MTP_BLOCKING_IND

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

MTP_MAINTENANCE_ORIENTED

MTP_HARDWARE_FAILURE_ORIENTED
: When one of these flags is set it indicates that either maintenance oriented or hardware failure oriented blocking is to be performed. If both or neither of these flags are set, the primitive will fail with error MBADFLAG.

Rules

b.1.8.8. MTP_BLOCKING_RES

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

MTP_MAINTENANCE_ORIENTED

MTP_HARDWARE_FAILURE_ORIENTED
: When one of these flags is set it indicates that either maintenance oriented or hardware failure oriented blocking is to be performed. If both or neither of these flags are set, the primitive will fail with error MBADFLAG.

Rules

b.1.8.9. MTP_BLOCKING_CON

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

MTP_MAINTENANCE_ORIENTED

MTP_HARDWARE_FAILURE_ORIENTED
: When one of these flags is set it indicates that either maintenance oriented or hardware failure oriented blocking is to be performed. If both or neither of these flags are set, the primitive will fail with error MBADFLAG.

Rules

b.1.8.10. MTP_UNBLOCKING_REQ

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

MTP_MAINTENANCE_ORIENTED

MTP_HARDWARE_FAILURE_ORIENTED
: When one of these flags is set it indicates that either maintenance oriented or hardware failure oriented blocking is to be performed. If both or neither of these flags are set, the primitive will fail with error MBADFLAG.

Rules

b.1.8.11. MTP_UNBLOCKING_IND

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

MTP_MAINTENANCE_ORIENTED

MTP_HARDWARE_FAILURE_ORIENTED
: When one of these flags is set it indicates that either maintenance oriented or hardware failure oriented blocking is

to be performed. If both or neither of these flags are set, the primitive will fail with error MBADFLAG.

Rules

b.1.8.12. MTP_UNBLOCKING_RES

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

MTP_MAINTENANCE_ORIENTED

MTP_HARDWARE_FAILURE_ORIENTED
When one of these flags is set it indicates that either maintenance oriented or hardware failure oriented blocking is to be performed. If both or neither of these flags are set, the primitive will fail with error MBADFLAG.

Rules

b.1.8.13. MTP_UNBLOCKING_CON

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

MTP_MAINTENANCE_ORIENTED

MTP_HARDWARE_FAILURE_ORIENTED
When one of these flags is set it indicates that either maintenance oriented or hardware failure oriented blocking is to be performed. If both or neither of these flags are set, the primitive will fail with error MBADFLAG.

Rules

b.1.8.14. MTP_QUERY_REQ

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

Rules

b.1.8.15. MTP_QUERY_IND

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

Rules

b.1.8.16. MTP_QUERY_RES

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

Rules

b.1.8.17. MTP_QUERY_CON

Parameters

mtp_flags:	Indicates the options flags. (See "Flags" below.)
mtp_addr_length:	Indicates the length of the address which consistes of a circuit identifier.
mtp_addr_offset:	Indicates the offset of the address from the start of the block.

Flags

MTP_GROUP
: When set, this flag indicates that the operation is to be performed on a group of MTP addresses and that any circuit identifier in the specified MTP address is to be interpreted by the MTP as a circuit group identifier.

Rules

c. Addendum for ETSI EN 300 356-1 V3.2.2 Conformace

This addendum describes the formats and rules that are specific to ETSI EN 300 356-1 V3.2.2. The addendum must be used along with the generic MTPI as defined in the main document, and the Q.764 conformance defined in Addendum 2, when implementing an MTP that will be configured with the EN 300 356-1 call processing layer.

c.1. Primitives and Rules for ETSI EN 300 356-1 V3.2.2 Conformance

The following are the additional rules that apply to the MTPI primitives for ETSI EN 300 356-1 V3.2.2 compatibility.

c.1.1. Local Management Primitives

c.1.2. Call Setup Primitives

c.1.2.1. MTP_SETUP_REQ

Parameters

Flags

Rules

c.1.2.2. MTP_SETUP_IND

Parameters

mtp_call_type: Specifies the call type to be set up. Also to Q.764 values, for EN 300 356-1 V3.2.2 conforming MTP providers, the call type can also be one of the values listed under "Call Type" below.

Call Type

The following call types are defined for EN 300 356-1 V3.2.2 conforming MTP providers in addition to the Q.931 values shown in Addendum 1.

MTP_CALL_TYPE_3x64KBS_UNRESTRICTED
: The call type is 3 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 3 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_4x64KBS_UNRESTRICTED
: The call type is 4 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 4 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_5x64KBS_UNRESTRICTED
: The call type is 5 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 5 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_6x64KBS_UNRESTRICTED
: The call type is 6 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of 384 kbit/s unrestricted digital information. This call type can be synonymous with MTP_CALL_TYPE_384KBS_UNRESTRICTED.

MTP_CALL_TYPE_7x64KBS_UNRESTRICTED
: The call type is 7 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 7 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_8x64KBS_UNRESTRICTED
: The call type is 8 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 8 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_9x64KBS_UNRESTRICTED
: The call type is 9 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 9 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_10x64KBS_UNRESTRICTED
: The call type is 10 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 10 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_11x64KBS_UNRESTRICTED
: The call type is 11 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 11 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_12x64KBS_UNRESTRICTED
: The call type is 12 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 12 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_13x64KBS_UNRESTRICTED
: The call type is 13 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 13 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_14x64KBS_UNRESTRICTED
: The call type is 14 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 14 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_15x64KBS_UNRESTRICTED
: The call type is 15 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 15 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_16x64KBS_UNRESTRICTED
: The call type is 16 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 16 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_17x64KBS_UNRESTRICTED
: The call type is 17 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 17 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_18x64KBS_UNRESTRICTED
: The call type is 18 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 28 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_19x64KBS_UNRESTRICTED
: The call type is 19 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 19 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_20x64KBS_UNRESTRICTED
: The call type is 20 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 20 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_21x64KBS_UNRESTRICTED
: This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 21 x 64 kbit/s unrestricted digital information". The call type is 21 x 64 kbit/s unrestricted digital information.

MTP_CALL_TYPE_22x64KBS_UNRESTRICTED
: The call type is 22 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 22 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_23x64KBS_UNRESTRICTED
: The call type is 23 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 23 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_24x64KBS_UNRESTRICTED
: The call type is 24 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "1536 kbit/s unrestricted digital information". This call type can be synonymous with MTP_CALL_TYPE_1536KBS_UNRESTRICTED.

MTP_CALL_TYPE_25x64KBS_UNRESTRICTED
: The call type is 25 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 25 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_26x64KBS_UNRESTRICTED
: The call type is 26 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 26 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_27x64KBS_UNRESTRICTED
: The call type is 27 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 27 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_28x64KBS_UNRESTRICTED
: The call type is 28 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "reserved for 28 x 64 kbit/s unrestricted digital information".

MTP_CALL_TYPE_29x64KBS_UNRESTRICTED
: The call type is 29 x 64 kbit/s unrestricted digital information. This call type corresponds to a EN 300 356-1 V3.2.2 transmission medium requirement of "1920 kbit/s unrestricted digital information". This call type can be synonymous with MTP_CALL_TYPE_1920KBS_UNRESTRICTED.

Rules

Rules for call type:

(1) Only multirate connection types for 384 kbit/s (6 x 64 kbit/s), 1536 kbit/s (24 x 64 kbit/s) and 1920 kbit/s (29 x 64 kbit/s) are supported. For EN 300 356-1 V3.2.2 compliant MTP providers.

A. Appendix A. Mapping of MTPI Primitives to Q.701

The mapping of MTPI primitives to Q.701 primitives is shown in *Table A-1*. For the most part, this mapping is a one to one mapping of service primitives, with the exception of *Connect Request* and *Disconnect Request*.

In Q.701 there is not concept of an "association" between MTP-entities. In OpenSS7 MTPI, the MTP_CONN_REQ and MTP_DISCON_REQ primitives are used to establish and release an association between MTP-entities.

Table A-1. Mapping of MTPI primitives to Q.701 Primitives

MTPI Primitive	*Q.701 Primitive*
MTP_INFO_REQ	−
MTP_INFO_ACK	−
MTP_BIND_REQ	−
MTP_BIND_ACK	−
MTP_UNBIND_REQ	−
MTP_ADDR_REQ	−
MTP_ADDR_ACK	−
MTP_OK_ACK	−
MTP_ERROR_ACK	−
MTP_CONN_REQ	−
MTP_DISCON_REQ	−
MTP_TRANSFER_REQ	MTP-TRANSFER Request
MTP_TRANSFER_IND	MTP-TRANSFER Indication
MTP_STATUS_IND	MTP-STATUS Indication
MTP_PAUSE_IND	MTP-PAUSE Indication
MTP_RESUME_IND	MTP-RESUME Indication
MTP_RESTART_BEGINS_IND	−
MTP_RESTART_COMPLETE_IND	−

B. Appendix B. Mapping of MTPI Primitives to ANSI T1.111.1

The mapping of MTPI primitives to T1.111.1 primitives is shown in *Table B-1*. For the most part, this mapping is a one to one mapping of service primitives, with the exception of *Connect Request* and *Disconnect Request*.

In T1.111.1 there is not concept of an “association” between MTP-entities. In OpenSS7 MTPI, the MTP_CONN_REQ and MTP_DISCON_REQ primitives are used to establish and release an association between MTP-entities.

Table B-1. Mapping of MTPI primitives to T1.111.1 Primitives

MTPI Primitive	*T1.111.1 Primitive*
MTP_INFO_REQ	−
MTP_INFO_ACK	−
MTP_BIND_REQ	−
MTP_BIND_ACK	−
MTP_UNBIND_REQ	−
MTP_ADDR_REQ	−
MTP_ADDR_ACK	−
MTP_OK_ACK	−
MTP_ERROR_ACK	−
MTP_CONN_REQ	−
MTP_DISCON_REQ	−
MTP_TRANSFER_REQ	MTP-TRANSFER Request
MTP_TRANSFER_IND	MTP-TRANSFER Indication
MTP_STATUS_IND	MTP-STATUS Indication
MTP_PAUSE_IND	MTP-PAUSE Indication
MTP_RESUME_IND	MTP-RESUME Indication
MTP_RESTART_BEGINS_IND	−
MTP_RESTART_COMPLETE_IND	−

C. Appendix C. State/Event Tables

D. Appendix D. Precedence Tables

E. Appendix E. MTPI Header File Listing

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

 @(#) $Id: sccpi.me,v 0.8.2.1 2003/07/29 00:34:46 brian Exp $

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

 Copyright (C) 2001-2002  OpenSS7 Corporation <http://www.openss7.com>
 Copyright (C) 1997-2000  Brian F. G. Bidulock <bidulock@dallas.net>

 All Rights Reserved.

 This program is free software; you can redistribute it and/or modify it under
 the terms of the GNU General Public License as published by the Free Software
 Foundation; either version 2 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 General Public License for more
 details.

 You should have received a copy of the GNU General Public License along with
 this program; if not, write to the Free Software Foundation, Inc., 675 Mass
 Ave, Cambridge, MA 02139, USA.

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

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

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

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

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

 Last Modified $Date: 2003/07/29 00:34:46 $ by $Author: brian $

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

#ifndef __SS7_MTPI_H__
#define __SS7_MTPI_H__

#ident "@(#) $Name:  $($Revision: 0.8.2.1 $) Copyright (c) 1997-2002 OpenSS7 Corporation."

#define MTP_VERSION_1       0x10
#define MTP_CURRENT_VERSION MTP_VERSION_1

typedef long mtp_long;
typedef unsigned long mtp_ulong;
typedef unsigned short mtp_ushort;
typedef unsigned char mtp_uchar;

#define MTP_BIND_REQ                    1       /* Bind to an MTP-SAP */
#define MTP_UNBIND_REQ                  2       /* Unbind from an MTP-SAP */
#define MTP_CONN_REQ                    3       /* Connect to a remote MTP-SAP */
#define MTP_DISCON_REQ                  4       /* Disconnect from a remote MTP-SAP */
#define MTP_ADDR_REQ                    5       /* Address service */
#define MTP_INFO_REQ                    6       /* Information service */
#define MTP_OPTMGMT_REQ                 7       /* Options management service */
#define MTP_TRANSFER_REQ                8       /* MTP data transfer request */

#define MTP_OK_ACK                      9       /* Positivie acknowledgement */
#define MTP_ERROR_ACK                   10      /* Negative acknowledgement */
#define MTP_BIND_ACK                    11      /* Bind acknowledgement */
#define MTP_ADDR_ACK                    12      /* Address acknowledgement */
#define MTP_INFO_ACK                    13      /* Information acknowledgement */
#define MTP_OPTMGMT_ACK                 14      /* Options management acknowledgement */
#define MTP_TRANSFER_IND                15      /* MTP data transfer indication */
#define MTP_PAUSE_IND                   16      /* MTP pause (stop) indication */
#define MTP_RESUME_IND                  17      /* MTP resume (start) indication */
#define MTP_STATUS_IND                  18      /* MTP status indication */
#define MTP_RESTART_COMPLETE_IND        19      /* MTP restart complete (impl. dep.) */

/*
 *  Interface States
 */
#define MTPS_UNBND                       0UL
#define MTPS_WACK_BREQ                   1UL
#define MTPS_IDLE                        2UL
#define MTPS_WACK_CREQ                   3UL
```

```
#define MTPS_WCON_CREQ                  4UL
#define MTPS_CONNECTED                  5UL
#define MTPS_WACK_UREQ                  6UL
#define MTPS_WACK_DREQ6                 7UL
#define MTPS_WACK_DREQ9                 8UL
#define MTPS_WACK_OPTREQ                9UL
#define MTPS_WREQ_ORDREL               10UL
#define MTPS_WIND_ORDREL               11UL
#define MTPS_WRES_CIND                 12UL
#define MTPS_UNUSABLE                  0xffffffffUL

#ifndef __HAVE_MTP_ADDR
#ifndef AF_MTP
#define AF_MTP 0
#endif
typedef struct mtp_addr {
    unsigned int family __attribute__ ((packed));
    unsigned short int ni __attribute__ ((packed));     /* network indentifier */
    unsigned short int si __attribute__ ((packed));     /* service indicator */
    unsigned int pc __attribute__ ((packed));   /* point code */
} mtp_addr_t;
#define __HAVE_MTP_ADDR
#endif

/*
 *  MTP_INFO_REQ, M_PROTO
 */
typedef struct MTP_info_req {
    mtp_ulong mtp_primitive;    /* always MTP_INFO_REQ */
} MTP_info_req_t;

/*
 *  MTP_INFO_ACK, M_PCPROTO
 */
typedef struct MTP_info_ack {
    mtp_ulong mtp_primitive;    /* always MTP_INFO_ACK */
    mtp_ulong mtp_msu_size;     /* maximum MSU size for guaranteed delivery */
    mtp_ulong mtp_addr_size;    /* maximum address size */
    mtp_ulong mtp_addr_length;  /* address length */
    mtp_ulong mtp_addr_offset;  /* address offset */
    mtp_ulong mtp_current_state;        /* current interface state */
    mtp_ulong mtp_serv_type;    /* service type */
    mtp_ulong mtp_version;      /* version of interface */
} MTP_info_ack_t;

#define M_COMS  1               /* Connection-mode MTP service supported */
#define M_CLMS  2               /* Connection-less MTP service supported */

/*
 *  MTP_ADDR_REQ, M_PCPROTO
 */
typedef struct MTP_addr_req {
    mtp_ulong mtp_primitive;    /* always MTP_ADDR_REQ */
} MTP_addr_req_t;

/*
 *  MTP_ADDR_ACK, M_PCPROTO
 */
typedef struct MTP_addr_ack {
    mtp_ulong mtp_primitive;    /* always MTP_ADDR_ACK */
    mtp_ulong mtp_loc_length;   /* length of local MTP address */
    mtp_ulong mtp_loc_offset;   /* offset of local MTP address */
    mtp_ulong mtp_rem_length;   /* length of remote MTP address */
    mtp_ulong mtp_rem_offset;   /* offset of remote MTP address */
} MTP_addr_ack_t;

/*
 *  MTP_BIND_REQ, M_PROTO
 */
typedef struct MTP_bind_req {
    mtp_ulong mtp_primitive;    /* always MTP_BIND_REQ */
    mtp_ulong mtp_addr_length;  /* length of MTP address */
    mtp_ulong mtp_addr_offset;  /* offset of MTP address */
    mtp_ulong mtp_bind_flags;   /* bind flags */
} MTP_bind_req_t;

/*
 *  MTP_BIND_ACK, M_PCPROTO
 */
typedef struct MTP_bind_ack {
    mtp_ulong mtp_primitive;    /* always MTP_BIND_ACK */
    mtp_ulong mtp_addr_length;  /* length of bound MTP address */
    mtp_ulong mtp_addr_offset;  /* offset of bound MTP address */
} MTP_bind_ack_t;

/*
 *  MTP_UNBIND_REQ, M_PROTO
 */
typedef struct MTP_unbind_req {
    mtp_ulong mtp_primitive;    /* always MTP_UNBIND_REQ */
} MTP_unbind_req_t;

/*
 *  MTP_CONN_REQ, M_PROTO
 */
typedef struct MTP_conn_req {
    mtp_ulong mtp_primitive;    /* always MTP_CONN_REQ */
    mtp_ulong mtp_addr_length;  /* length of MTP address to connect */
    mtp_ulong mtp_addr_offset;  /* offset of MTP address to connect */
    mtp_ulong mtp_conn_flags;   /* connect flags */
} MTP_conn_req_t;

/*
 *  MTP_DISCON_REQ, M_PROTO, M_PCPROTO
 */
typedef struct MTP_discon_req {
    mtp_ulong mtp_primitive;    /* always MTP_DISCON_REQ */
} MTP_discon_req_t;

/*
 *  MTP_OPTMGMT_REQ, M_PROTO or M_PCPROTO
 */
typedef struct MTP_optmgmt_req {
    mtp_ulong mtp_primitive;    /* always MTP_OPTMGMT_REQ */
    mtp_ulong mtp_opt_length;   /* length of options */
    mtp_ulong mtp_opt_offset;   /* offset of options */
    mtp_ulong mtp_mgmt_flags;   /* management flags */
} MTP_optmgmt_req_t;

#define MTP_DEFAULT     0UL
#define MTP_CHECK       1UL
#define MTP_NEGOTIATE   2UL
#define MTP_CURRENT     3UL

/*
 *  MTP_OPTMGMT_ACK, M_PCPROTO
 */
typedef struct MTP_optmgmt_ack {
    mtp_ulong mtp_primitive;    /* always MTP_OPTMGMT_ACK */
    mtp_ulong mtp_opt_length;   /* length of options */
    mtp_ulong mtp_opt_offset;   /* offset of options */
    mtp_ulong mtp_mgmt_flags;   /* management flags */
} MTP_optmgmt_ack_t;

/*
 *  MTP_OK, MTP_ERROR, M_PCPROTO
 */
typedef struct MTP_ok_ack {
    mtp_ulong mtp_primitive;    /* always MTP_OK_ACK */
    mtp_ulong mtp_correct_prim; /* correct primitive */
} MTP_ok_ack_t;

typedef struct MTP_error_ack {
    mtp_ulong mtp_primitive;    /* always MTP_ERROR_ACK */
    mtp_ulong mtp_error_primitive;      /* primitive in error */
    mtp_ulong mtp_mtpi_error;   /* MTP interface error */
    mtp_ulong mtp_unix_error;   /* UNIX error */
} MTP_error_ack_t;

#define MSYSERR         0UL
#define MACCESS         1UL
#define MBADADDR        2UL
#define MNOADDR         3UL
#define MBADPRIM        4UL
#define MOUTSTATE       5UL
#define MNOTSUPP        6UL
#define MBADFLAG        7UL
#define MBADOPT         8UL

/*
 *  MTP_TRANSFER_REQ, M_PROTO
 */
typedef struct MTP_transfer_req {
    mtp_ulong mtp_primitive;    /* always MTP_TRANSFER_REQ */
    mtp_ulong mtp_dest_length;  /* length of destination address */
    mtp_ulong mtp_dest_offset;  /* offset of destination address */
    mtp_ulong mtp_mp;           /* message priority */
    mtp_ulong mtp_sls;          /* signalling link selection */
} MTP_transfer_req_t;

/*
 *  MTP_TRANSFER_IND, M_PROTO
 */
typedef struct MTP_transfer_ind {
    mtp_ulong mtp_primitive;    /* always MTP_TRANSFER_IND */
    mtp_ulong mtp_srce_length;  /* length of source address */
    mtp_ulong mtp_srce_offset;  /* offset of source address */
    mtp_ulong mtp_mp;           /* message priority */
    mtp_ulong mtp_sls;          /* signalling link selection */
} MTP_transfer_ind_t;

/*
 *  MTP_PAUSE_IND, M_PCPROTO
 */
typedef struct MTP_pause_ind {
    mtp_ulong mtp_primitive;    /* always MTP_PAUSE_IND */
    mtp_ulong mtp_addr_length;  /* length of affected MTP address */
    mtp_ulong mtp_addr_offset;  /* offset of affected MTP address */
} MTP_pause_ind_t;

/*
 *  MTP_RESUME_IND, M_PCPROTO
 */
typedef struct MTP_resume_ind {
    mtp_ulong mtp_primitive;    /* always MTP_RESUME_IND */
    mtp_ulong mtp_addr_length;  /* length of affected MTP address */
    mtp_ulong mtp_addr_offset;  /* offset of affected MTP address */
} MTP_resume_ind_t;

/*
 *  MTP_STATUS_IND, M_PCPROTO
 */
typedef struct MTP_status_ind {
    mtp_ulong mtp_primitive;    /* always MTP_STATUS_IND */
    mtp_ulong mtp_addr_length;  /* length of affected MTP address */
    mtp_ulong mtp_addr_offset;  /* offset of affected MTP address */
    mtp_ulong mtp_type;         /* type */
    mtp_ulong mtp_status;       /* status */
} MTP_status_ind_t;

/* Type for MTP_STATUS_IND */
#define MTP_STATUS_TYPE_CONG            0x00    /* MTP-STATUS refers to congestion */
#define MTP_STATUS_TYPE_UPU             0x01    /* MTP-STATUS referes to user part unavailability */

/* Status for MTP_STATUS_IND, with MTP_STATUS_TYPE_UPU */
#define MTP_STATUS_UPU_UNKNOWN          0x01    /* User part unavailable: unknown */
#define MTP_STATUS_UPU_UNEQUIPPED       0x02    /* User part unavailable: unequipped remote user. */
#define MTP_STATUS_UPU_INACCESSIBLE     0x03    /* User part unavailable: inaccessible remote user.
                                                 */

/* Status for MTP_STATUS_IND, with MTP_STATUS_TYPE_CONG */
#define MTP_STATUS_CONGESTION_LEVEL0    0x00    /* Signalling network congestion level 0 */
#define MTP_STATUS_CONGESTION_LEVEL1    0x01    /* Signalling network congestion level 1 */
#define MTP_STATUS_CONGESTION_LEVEL2    0x02    /* Signalling network congestion level 2 */
#define MTP_STATUS_CONGESTION_LEVEL3    0x03    /* Signalling network congestion level 3 */
#define MTP_STATUS_CONGESTION           0x04    /* Signalling network congestion */

/*
 *  MTP_RESTART_COMPLETE_IND, M_PCPROTO
 */
typedef struct MTP_restart_complete_ind {
    mtp_ulong mtp_primitive;    /* always MTP_RESTART_COMPLETE_IND */
} MTP_restart_complete_ind_t;

union MTP_primitives {
    mtp_ulong mtp_primitive;
    MTP_info_req_t info_req;
    MTP_info_ack_t info_ack;
    MTP_addr_req_t addr_req;
    MTP_addr_ack_t addr_ack;
    MTP_bind_req_t bind_req;
    MTP_bind_ack_t bind_ack;
    MTP_unbind_req_t unbind_req;
    MTP_conn_req_t conn_req;
    MTP_discon_req_t discon_req;
    MTP_optmgmt_req_t optmgmt_req;
    MTP_optmgmt_ack_t optmgmt_ack;
    MTP_ok_ack_t ok_ack;
    MTP_error_ack_t error_ack;
    MTP_transfer_req_t transfer_req;
    MTP_transfer_ind_t transfer_ind;
    MTP_pause_ind_t pause_ind;
    MTP_resume_ind_t resume_ind;
    MTP_status_ind_t status_ind;
    MTP_restart_complete_ind_t restart_complete_ind;
};

typedef struct {
    mtp_ulong mtp_affected_dpc;
} mtp_pause_ind_t;

typedef struct {
    mtp_ulong mtp_affected_dpc;
} mtp_resume_ind_t;

/*
 *  8.1 Transfer
 *
 *  The primitive "MTP-TRANSFER" is used between level 4 and level 3 (SMH) to
 *  provide the MTP message transfer service.
 */

/*
 *  8.2 Pause
 *
 *  The primitive "MTP-PAUSE" indicates to "Users" the total inability of
 *  providing the MTP service to the specified destination (see 7.2.6).
 *
 *  NOTE -  The signalling point is inacessible via the MTP.  The MTP will
 *  determine when the signalling point is again acessible and send MTP-RESUME
 *  indication.  The user should wait for such an indication and, meanwhile is
 *  not allowed to send messages on that signalling point.  If the remote peer
 *  user is thought to be unavailable, that condition may be maintained or
 *  cancelled at the local user’s discretion.
 */

/*
 *  8.3 Resume
 *
 *  The primitive MTP-RESUME indications to the "User" the ability of
 *  providing the MTP service to the specified destination (See 7.2.6)
```

```
 *
 *  This primitive corresponds to the destination accessible state as defined
 *  in Recommendation Q.704.
 *
 *  NOTE -  When the MTP-RESUME indicaiton is given to each user, the MTP does
 *          not know whether the remote peer user is available.  This is the
 *          responsibility of each user.
 */

/*
 *  8.4 Status
 *
 *  The primitive "MTP-STATUS" indicates to the "Users" the partial inability
 *  of providing the MTP service to the specified destination.  The primitive
 *  is also used to indicate to a User that a remote corresponding User is
 *  unavailable and the cause for unavailability (see 11.2.7/Q.704).
 *
 *  In the case of national option with congestion priorities or multiple
 *  signalling link congestion states without prioritites as in Recommendation
 *  Q.704 are implemented, this "MTP-STATUS" primitive is also used to
 *  indicate a change of congestion level.
 *
 *  This primitive corresponds to the destination congested/User Part
 *  unavailable states as defined in Recommendation Q.704.
 *
 *  NOTE -  In the case of remote user unavailability, the user is responsible
 *          for determining the availability of this peer user.  The user is
 *          cautioned not to send normal traffic to the peer user because,
 *          while such peer is unavailable, no message will be delivered but
 *          each will result in a repeated "MTP-STATUS" indication.  The MTP
 *          will not send any further indications about the unavailability or
 *          availability of this peer user unless the local user continues to
 *          send messages to the peer user.
 */

/*
 *  8.5 Restart
 *
 *  When the MTP restart procedure is terminated, the MTP indicates the end of
 *  MTP restart to all local MTP Users showing each signalling point’s
 *  accessibility or inaccessibility.  The means of doing this is
 *  implementation dependent (see 9/Q.704).
 */

/*
 *  MTP_STATUS_IND, M_PROTO or M_PCPROTO
 */
typedef struct {
    mtp_ulong mtp_affected_dpc;
    mtp_uchar mtp_cause;
    mtp_uchar mtp_level;
} mtp_status_ind_t;

typedef struct {
    mtp_ulong dpc;
    mtp_ulong opc;
    mtp_ulong sls;
} mtp_rl_t;

typedef struct {
    mtp_uchar si;
    mtp_uchar mp;
    mtp_uchar ni;
    mtp_rl_t rl;
} mtp_hdr_t;

typedef struct {
    mtp_uchar si;
    mtp_uchar mp;
    mtp_uchar ni;
    mtp_rl_t rl;
    mtp_uchar h0;
    mtp_uchar h1;
} mtp_msu_t;

typedef struct {
    mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
    mtp_ulong mtp_signal;       /* MTP_SIGNAL_COO */
    mtp_msu_t mtp_msg;
    mtp_ulong mtp_slc;
    mtp_ulong mtp_fsnc;
} mtp_signal_coo_t;

typedef struct {
    mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
    mtp_ulong mtp_signal;       /* MTP_SIGNAL_COA */
    mtp_msu_t mtp_msg;
} mtp_signal_coa_t;

typedef struct {
    mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
    mtp_ulong mtp_signal;       /* MTP_SIGNAL_CBD */
    mtp_msu_t mtp_msg;
} mtp_signal_cbd_t;

typedef struct {
    mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
```

```
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_CBA */
        mtp_msu_t mtp_msg;
    } mtp_signal_cba_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_ECO */
        mtp_msu_t mtp_msg;
    } mtp_signal_eco_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_ECA */
        mtp_msu_t mtp_msg;
    } mtp_signal_eca_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_RCT */
        mtp_msu_t mtp_msg;
    } mtp_signal_rct_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_TFC */
        mtp_msu_t mtp_msg;
    } mtp_signal_tfc_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_TFP */
        mtp_msu_t mtp_msg;
    } mtp_signal_tfp_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_TFR */
        mtp_msu_t mtp_msg;
    } mtp_signal_tfr_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_TFA */
        mtp_msu_t mtp_msg;
    } mtp_signal_tfa_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_RSP */
        mtp_msu_t mtp_msg;
    } mtp_signal_rsp_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_RSR */
        mtp_msu_t mtp_msg;
    } mtp_signal_rsr_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_LIN */
        mtp_msu_t mtp_msg;
    } mtp_signal_lin_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_LUN */
        mtp_msu_t mtp_msg;
    } mtp_signal_lun_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_LIA */
        mtp_msu_t mtp_msg;
    } mtp_signal_lia_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_LUA */
        mtp_msu_t mtp_msg;
    } mtp_signal_lua_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_LID */
        mtp_msu_t mtp_msg;
    } mtp_signal_lid_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_LFU */
        mtp_msu_t mtp_msg;
    } mtp_signal_lfu_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_LLT */
        mtp_msu_t mtp_msg;
    } mtp_signal_llt_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_LRT */
        mtp_msu_t mtp_msg;
    } mtp_signal_lrt_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_TRA */
        mtp_msu_t mtp_msg;
    } mtp_signal_tra_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_DLC */
        mtp_msu_t mtp_msg;
    } mtp_signal_dlc_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_CSS */
        mtp_msu_t mtp_msg;
    } mtp_signal_css_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_CNS */
        mtp_msu_t mtp_msg;
    } mtp_signal_cns_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_CNP */
        mtp_msu_t mtp_msg;
    } mtp_signal_cnp_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_UPU */
        mtp_msu_t mtp_msg;
    } mtp_signal_upu_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_SLTM */
        mtp_msu_t mtp_msg;
    } mtp_signal_sltm_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_SLTA */
        mtp_msu_t mtp_msg;
    } mtp_signal_slta_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_SSLTM */
        mtp_msu_t mtp_msg;
    } mtp_signal_ssltm_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_SSLTA */
        mtp_msu_t mtp_msg;
    } mtp_signal_sslta_t;

    typedef struct {
        mtp_long mtp_primitive;     /* MTP_MSU_REQ, MTP_MSU_IND */
        mtp_ulong mtp_signal;       /* MTP_SIGNAL_USER */
        mtp_hdr_t mtp_msg;
    } mtp_signal_user_t;

    typedef union {
        mtp_long mtp_primitive;
        mtp_signal_user_t msg;
        mtp_signal_coo_t coo;
        mtp_signal_coa_t coa;
        mtp_signal_cbd_t cbd;
        mtp_signal_cba_t cba;
        mtp_signal_eco_t eco;
        mtp_signal_eca_t eca;
        mtp_signal_rct_t rct;
        mtp_signal_tfc_t tfc;
        mtp_signal_tfp_t tfp;
        mtp_signal_tfr_t tfr;
        mtp_signal_tfa_t tfa;
        mtp_signal_rsp_t rsp;
        mtp_signal_rsr_t rsr;
        mtp_signal_lin_t lin;
        mtp_signal_lun_t lun;
        mtp_signal_lia_t lia;
        mtp_signal_lua_t lua;
        mtp_signal_lid_t lid;
        mtp_signal_lfu_t lfu;
        mtp_signal_llt_t llt;
        mtp_signal_lrt_t lrt;
        mtp_signal_tra_t tra;
        mtp_signal_dlc_t dlc;
        mtp_signal_css_t css;
        mtp_signal_cns_t cns;
        mtp_signal_cnp_t cnp;
        mtp_signal_upu_t upu;
        mtp_signal_sltm_t sltm;
        mtp_signal_slta_t slta;
        mtp_signal_ssltm_t ssltm;
        mtp_signal_sslta_t sslta;
        mtp_signal_user_t user;
    } MTP_signals;

    /*
     *  MTP_MSU_REQ , M_PROTO or M_PCPROTO (M_DATA)
     */
    typedef MTP_signals mtp_msu_req_t;
    /*
     *  MTP_MSU_IND , M_PROTO or M_PCPROTO (M_DATA)
     */
    typedef MTP_signals mtp_msu_ind_t;

    #define MTP_SIGNAL_NONE          0
    #define MTP_SIGNAL_COO           1      /* STM */
    #define MTP_SIGNAL_COA           2      /* STM */
    #define MTP_SIGNAL_CBD           3      /* STM */
    #define MTP_SIGNAL_CBA           4      /* STM */
    #define MTP_SIGNAL_ECO           5      /* STM */
    #define MTP_SIGNAL_ECA           6      /* STM */
    #define MTP_SIGNAL_LIN          14      /* STM */
    #define MTP_SIGNAL_LUN          15      /* STM */
    #define MTP_SIGNAL_LIA          16      /* STM */
    #define MTP_SIGNAL_LUA          17      /* STM */
    #define MTP_SIGNAL_LID          18      /* STM */
    #define MTP_SIGNAL_LFU          19      /* STM */
    #define MTP_SIGNAL_LLT          20      /* STM */
    #define MTP_SIGNAL_LRT          21      /* STM */
    #define MTP_SIGNAL_TRA          22      /* STM */

    #define MTP_SIGNAL_RCT           7      /* SRM */
    #define MTP_SIGNAL_TFC           8      /* SRM */
    #define MTP_SIGNAL_TFP           9      /* SRM */
    #define MTP_SIGNAL_TFR          10      /* SRM */
    #define MTP_SIGNAL_TFA          11      /* SRM */
    #define MTP_SIGNAL_RSP          12      /* SRM */
    #define MTP_SIGNAL_RSR          13      /* SRM */
    #define MTP_SIGNAL_UPU          27      /* SRM */

    #define MTP_SIGNAL_DLC          23      /* SLM */
    #define MTP_SIGNAL_CSS          24      /* SLM */
    #define MTP_SIGNAL_CNS          25      /* SLM */
    #define MTP_SIGNAL_CNP          26      /* SLM */

    #define MTP_SIGNAL_SLTM         28      /* SLTC */
    #define MTP_SIGNAL_SLTA         29      /* SLTC */
    #define MTP_SIGNAL_SSLTM        30      /* SLTC */
    #define MTP_SIGNAL_SSLTA        31      /* SLTC */

    #define MTP_SIGNAL_USER         32      /* L4 */

    #endif                          /* __SS7_MTPI_H__ */
```

References

NPI. UNIX. International, “Network Provider Interface Specification,” NPI Revision 2.0.0, UNIX International Publication, Parsippany, New Jersey (August 17, 1992). http://www.openss7.org/doc/npi.pdf

TPI. Open Group, “Transport Provider Interface Specification,” TPI Version 2, Draft 2, Open Group Publication (1999). http://www.opengroup.org/onlinepubs/

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

Q.704.
ITU, “Message Transfer Part − Signalling Network Functions and Messages,” ITU-T Recommendation Q.704, ITU-T Telecommunication Standardization Sector of ITU, Geneva (March 1993). (Previously "CCITT Recommendation")

T1.111.
ANSI, “Signalling System No. 7 − Message Transfer Part,” ANSI T1.111, American National Standards Institue (1992).

SVID. *System V, Interface Definition*

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

Q.764.
ITU, “Signalling System No. 7 − ISDN User Part Signalling Procedures,” ITU-T Recommendation Q.764, ITU-T Telecommunication Standardization Sector of ITU, Geneva (March 1993). (Previously "CCITT Recommendation")

Q.714.
ITU, “Signalling Connection Control Part Procedures,” ITU-T Recommendation Q.714, ITU-T Telecommunication Standardization Sector of ITU, Geneva (March 1993). (Previously "CCITT Recommendation")

Q.711.
ITU, “Functional Description of Signalling Connection Control Part,” ITU-T Recommendation Q.711, ITU-T Telecommunication Standardization Sector of ITU, Geneva (March 1993). (Previously "CCITT Recommendation")

Q.724.
ITU, “Signalling System No. 7 − Telephone User Part − Signalling Procedures,” ITU-T Recommendation Q.724, ITU-T Telecommunication Standardization Sector of ITU, Geneva (November 1988). (Previously "CCITT Recommendation")

M3UA.
G. Sidebottom, K. Morneault, J. Pastor-Balbas, (eds), “Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) - User Adaptation Layer (M3UA),” RFC 3332, Internet Engineering Task Force - Signalling Transport Working Group (September, 2002).

X.210.
ITU, “Basic Reference Model: Conventions for the Defintion of OSI Services,” ITU-T Recommendation X.210, ITU-T Telecommunication Standardization Sector of ITU, Geneva (November 1993). (Previously "CCITT Recommendation")

JT-Q.704.
TTC, “Message Transfer Part − Signalling Network Functions and Messages,” TTC Standard JT-Q.704, Telecommunication Technology Committee (TTC) (April 28, 1992).

List of Illustrations

List of Tables

Table of Contents