Transaction Interface (TRI) Specification

Transaction Interface (TRI) Specification

Version 0.9a Edition 8
Updated 2008-10-31
Distributed with Package strss7-0.9a.8

Abstract

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

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

Published by:

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

Notice:

Short Contents

Table of Contents

List of Figures

List of Tables

Preface

Security Warning

Abstract

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

This document specifies a Transaction Interface (TRI) Specification in support of the OpenSS7 Transaction Capablities Application Part (TCAP) protocol stacks. It provides abstraction of the transaction interface to these components as well as providing a basis for transaction control for other transaction control protocols.

Purpose

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

Intent

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

Audience

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

Disclaimer

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

Revision History

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

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

```
tri.texi,v
Revision 0.9.2.20  2008-09-20 11:04:33  brian
- added package patchlevel

Revision 0.9.2.19  2008-08-03 06:03:33  brian
- protected agains texinfo commands in log entries

Revision 0.9.2.18  2008-08-03 05:05:18  brian
- conditional @syncodeindex frags out automake, fails distcheck

Revision 0.9.2.17  2008-07-11 09:36:14  brian
- updated documentation

Revision 0.9.2.16  2008-04-29 07:10:40  brian
- updating headers for release

Revision 0.9.2.15  2007/08/03 13:34:59  brian
- manual updates, put ss7 modules in public release
```

1 Introduction

This document specifies a *STREAMS*-based kernel-level instantiation of the ITU-T Transaction Capabilities Application Part (TCAP) Transaction (TR) Sub-Layer. The Transaction Interface (TRI) enables the user of a transaction sub-layer service to access and use any of a variety of conforming transaction providers without specific knowledge of the provider's protocol. The service interface is designed to support any transaction protocol. This interface only specifies access to transaction sub-layer services providers, and does not address issues concerning transaction sub-layer management, protocol performance, and performance analysis tools.

The specification assumes that the reader is familiar with the ISO reference model terminology, ISO/ITU-T transaction service definitions (ROSE, ACSE, TCAP), and STREAMS.

1.1 Related Documentation

— ITU-T Recommendation X.200 (White Book) — ISO/IEC 7498-1:1994
— ITU-T Recommendation X.219 (White Book) — ISO/IEC
— ITU-T Recommendation X.229 (White Book) — ISO/IEC
— ITU-T Recommendation X.217 (White Book) — ISO/IEC 8649 : 1996
— ITU-T Recommendation X.227 (White Book) — ISO/IEC 8650-1 : 1995
— ITU-T Recommendation X.237 (White Book) — ISO/IEC 10035-1 : 1995
— ITU-T Recommendation Q.771 (White Book)
— System V Interface Definition, Issue 2 - Volume 3

1.1.1 Role

This document specifies an interface that supports the service provided by the Association Control Service Element (ACSE) for Open Systems Interconnect for ITU-T Applications as specified in ITU-T Recommendation X.217 (ISO/IEC 8649). It is also intended to support the Transaction Sub-layer provided by the Transaction Capabilities Application Part (TCAP) for Signalling System Number 7 (SS7) as specified in ITU-T Recommendation Q.771. These specifications are targeted for use by developers and testers of protocol modules that require transaction sub-layer service.[1]

1.2 Definitions, Acronyms, and Abbreviations

Originating TR User
: A TR-User that initiates a transaction.

Destination TR User
: A TR-User with whom an originating TR user wishes to establish a transaction.

ISO
: International Organization for Standardization

[1] For an alternative interface, see Section "Introduction" in *Transaction Component Interface*, or Section "Introduction" in *Using XTI for TCAP*.

TR User Kernel level protocol or user level application that is accessing the services of the transaction sub-layer.

TR Provider
Transaction sub-layer entity/entities that provide/s the services of the transaction interface.

TRI Transaction Interface

TIDU Transaction Interface Data Unit

TSDU Transaction Service Data Unit

OSI Open Systems Interconnection

QOS Quality of Service

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

2 The Transaction Sub-Layer

The Transaction Sub-Layer provides the means to manage the association of TR-User into transactions. It is responsible for the routing and management of transaction associations between TR-user entities.

2.1 Model of the TRI

The TRI defines the services provided by the transaction sub-layer to the transaction-user at the boundary between the Transaction Component (TC) Sub-Layer and the Transaction (TR) Sub-Layer in the model presented in ITU-T Recommendation Q.771. The interface consists of a set of primitives defined as STREAMS messages that provide access to the transaction sub-layer services, and are transferred between the TR user entity and the TR provider. These primitives are of two types: ones that originate from the TR user, and others that originate from the TR provider, or respond to an event of the TR provider. The primitives that originate from the TR provider are either confirmations of a request or are indications to the NS user that the event has occurred. Figure 2.1 shows the model of the TRI.

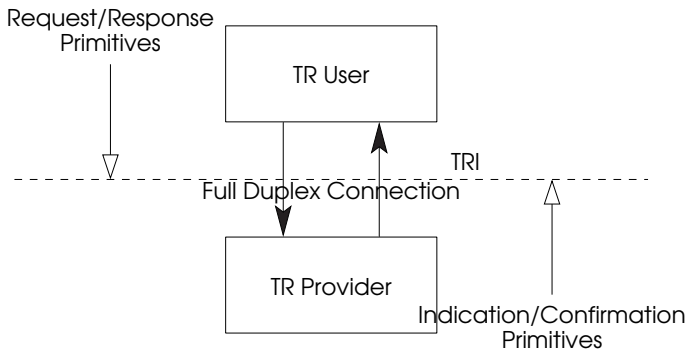

Figure 2.1: *Model of the TRI*

The TRI allows the TR provider to be configured with any transaction sub-layer user (such as the Transaction Component (TC) Sub-Layer) that also conforms to the TRI. A transaction sub-layer user can also be a user program that conforms to the TRI and accesses the TR provider via `putmsg(2s)` and `getmsg(2s)` system calls.

STREAMS messages that are used to communicate transaction service primitives between the transaction user and the transaction provider may have one of the following formats:

1. A `M_PROTO` message block followed by zero or more `M_DATA` message blocks. The `M_PROTO` message block contains the type of service primitive and all relevant arguments associated with the primitive. The `M_DATA` blocks contain user data associated with the service primitive.

2. One `M_PCPROTO` message block containing the type of service primitive and all the relevant arguments associated with the primitive.
3. One or more `M_DATA` message blocks containing user data.

The following sections describe the service primitives which define both connection-mode and connectionless-mode service.

For both types of service, two types of primitives exist: primitives that originate from the service user and primitives that originate from the service provider. The primitives that originate from the service user make requests to the service provider or response to an event of the service provider. The primitive that originate from the service provider are either confirmations of a request or are indications to the service user that an event has occurred. The primitive types along with the mapping of those primitives to the *STREAMS* message types and the service primitives of the ISO/IEC xxxxx and service definitions are listed in Chapter 4 [TRI Primitives], page 21. The format of these primitives and the rules governing the use of them are described in Section 4.1 [Management Primitives], page 22, Section 4.2 [Connection-Oriented Mode Primitives], page 45, and Section 4.3 [Connectionless Mode Primitives], page 68.

2.2 TRI Services

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

The services supported by the TRI are based on two distinct modes of communication, connection-mode transaction service (COTS) and connectionless transaction service (CLTS). Also, the TRI supports services for local management.

2.2.1 COTS

The main features of the connection mode communication are:

a. It is virtual circuit oriented;

b. it provides transfer of data via a pre-established path; and,

c. it provides reliable data transfer.[1]

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

2.2.2 CLTS

The main features of the connectionless mode communication are:

a. It is datagram oriented;

b. it provides transfer of data in self contained units;

[1] That is, it supports TCAP
operation classes 1, 2, and 3; ROSE operation classes 1, 2, 3 and 4.

c. there is no logical relationship between these units of data; and,

d. it is unreliable.

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. As the units of data are transmitted independently from source to destination, there are, in general, no guarantees of proper sequence and completeness of the data stream.

2.2.3 Local Management

The TRI specifications also define a set of local management functions that apply to both COTS and CLTS modes of communication. These services have local significance only.

Table 1 and Table 2 summarizes the TRI service primitives by their state and service.

STATE	SERVICE	PRIMITIVES
Local Management	Information Reporting	TR_INFO_REQ, TR_INFO_ACK, TR_ERROR_ACK
	Bind	TR_BIND_REQ, TR_BIND_ACK, TR_UNBIND_ACK, TR_OK_ACK, TR_ERROR_ACK
	Options Management	TR_OPTMGMT_REQ, TR_OK_ACK, TR_ERROR_ACK
Transaction Establishment	Transaction Begin	TR_BEGIN_REQ, TR_BEGIN_IND, TR_BEGIN_RES, TR_BEGIN_CON, TR_TOKEN_REQ, TR_TOKEN_ACK, TR_OK_ACK, TR_ERROR_ACK
Transaction Data Transfer	Transaction Continue	TR_CONT_REQ, TR_CONT_IND
Transaction Release	Transaction End	TR_END_REQ, TR_END_IND
	Transaction Abort	TR_ABORT_REQ, TR_ABORT_IND

Table 1. *Service Primitives for Connection Mode Transaction*

STATE	SERVICE	PRIMITIVES
Local Management	Information Reporting	TR_INFO_REQ, TR_INFO_ACK, TR_ERROR_ACK
	Bind	TR_BIND_REQ, TR_BIND_ACK, TR_UNBIND_ACK, TR_OK_ACK, TR_ERROR_ACK
	Options Management	TR_OPTMGMT_REQ, TR_OK_ACK, TR_ERROR_ACK
Transaction Unitdata	Transaction Unidirectional	TR_UNI_REQ, TR_UNI_IND, TR_NOTICE_IND

Table 2. *Service Primitives for Connectionless Mode Transaction*

3 TRI Services Definition

This section describes the services of the TRI primitives. Time-sequence diagrams[1] that illustrate the sequence of primitives are used. 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 the international standards. These services apply to both connection-mode as well as connectionless modes of communication. They are involved for the initialization/de-initialization of a stream connected to the TR provider. They are also used to manage options supported by the TR provider and to report information on the supported parameter values.

3.1.1 Transaction Information Reporting Service

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

- `TR_INFO_REQ`: This primitive request that the TR provider returns the values of all the supported protocol parameters. This request may be invoked during any phase.
- `TR_INFO_ACK`: This primitive is in response to the *TR_INFO_REQ* primitive and returns the values of the supported protocol parameters to the TR user.

The sequence of primitives for transaction information management is shown in Figure 3.1.

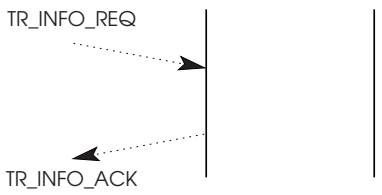

Figure 3.1: *Sequence of Primitives – Transaction Information Reporting Service*

3.1.2 TR User Bind Service

This service allows an originating address to be associated with a stream. It allows the TR user to negotiate the number of transaction begin indications that can remain unacknowledged for that TR user (a transaction begin indication is considered unacknowledged while it is awaiting a corresponding transaction response or abort request from the TR user). This service also defines a mechanism that allows a stream (bound to the address of the TR user) to be reserved to handle incoming transactions only. This stream is referred to as the listener stream.

- `TR_BIND_REQ`: This primitive request that the TR user be bound to a particular originating address, and negotiate the number of allowable outstanding transaction indications for that address.

[1] Conventions for the time-sequence diagrams are defined in ITU-T X.210, ISO/IEC 10731:1994.

- `TR_BIND_ACK`: This primitive is in response to the `TR_BIND_REQ` primitive and indicates to the user that the specified TR user has been bound to a protocol address.

The sequence of primitives for the TR user bind service is shown in Figure 3.2.

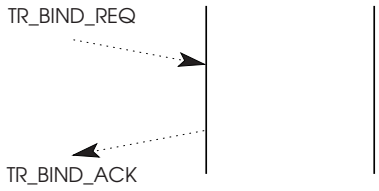

Figure 3.2: *Sequence of Primitives – TR User Bind Service*

3.1.3 TR User Unbind Service

This service allows the TR user to be unbound from a protocol address.

- `TR_UNBIND_REQ`: This primitive requests that the TR user be unbound from the protocol address it had previously been bound to.

The sequence of primitives for the TR user unbind service is shown in Figure 3.3.

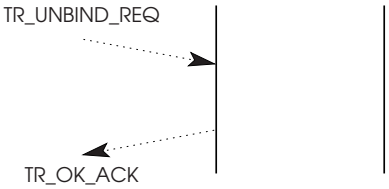

Figure 3.3: *Sequence of Primitives – TR User Unbind Receipt Acknowledgement Services*

3.1.4 Receipt Acknowledgement Service

- `TR_OK_ACK`: This primitive indicates to the TR user that the previous TR user originated primitive was received successfully by the TR provider.

An example showing the sequence of primitives for successful receive acknowledgement is depicted in Figure 3.3.

3.1.5 Options Mangement Service

This service allows the TR user to manage the QOS parameter values associated with the TR provider.

- `TR_OPTMGMT_REQ`: This primitive allows the TR user to select default values for QOS parameters within the range supported by the TR provider, and to indicate the default selection of return option.

Figure 3.4 shows the sequence of primitives for transaction options management.

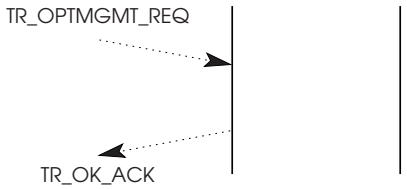

Figure 3.4: *Sequence of Primitives – Options Management Service*

3.1.6 Error Acknowledgement Service

- `TR_ERROR_ACK`: This primitive indicates to the TR user that a non-fatal error has occurred in the last TR user originated request or response primitive (listed in Figure 3.5) on the stream.

Figure 3.5 shows the sequence of primitives for the error management primitive.

Figure 3.5: *Sequence of Primitives – Error Acknowledgement Service*

3.2 Connection-Oriented Mode Services Definition

This section describes the required transaction service primitives that define the connection mode interface.

The queue model for connection-oriented services are discussed in more detail in ITU-T X.217 and ITU-T Q.771.

The queue model represents the operation of a transaction association in the abstract by a pair of queues linking two transaction users. There is one queue for each direction of data flow. Each queue represents a flow control function in one direction of transfer. The ability of a user to add objects to a queue will be determined by the behaviour 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 transaction association. Objects that are entered or removed from the queue are either as a result of interactions at the two transaction addresses, or as the result of TR provider initiatives.

- A queue is empty until a transaction object has been entered and can be returned to this state, with loss of its contents, by the TR provider.
- Objects may be entered into a queue as a result of the actions of the source TR user, subject to control by the TR provider.
- Objects may also be entered into a queue by the TR provider.

- Objects are removed from the queue under the control of the TR user in the same order as they were entered except:
  1. If the object is of 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
  2. 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, "abort" objects are defined to be destructive with respect to all other objects.

Table 3 shows the ordering relationships among the queue model objects.

Object X / **Object Y**	**BEGIN**	**CONT**	**END**	**ABORT**
BEGIN	N/A	–	–	DES
CONT	N/A	–	–	DES
END	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 the 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.

Table 3. *Ordering Relationships Between Queue Model Objects*

3.2.1 Transaction Initiation Phase

A pair of queues is associated with a transaction association between two transaction users when the TR provider receives a `TR_BEGIN_REQ` primitive at one of the TR users resulting in a begin object being entered into the queue. The queues will remain associated with the transaction until a `TR_END_REQ` or `TR_ABORT_REQ` primitive (resulting in an end or abort object) is either entered or removed from a queue. Similarly, in the queue from the destination TR user, objects can be entered into the queue only after the begin object associated with the `TR_BEGIN_RES` has been entered into the queue. Alternatively, the destination TR user can enter an end or abort object into the queue instead of the begin object to terminate the transaction.

The transaction establishment procedure will fail if the TR provider is unable to establish a transaction association, or if the destination TR user is unable to accept the `TR_BEGIN_IND` (see Transaction Termination primitive definition in Section 4.2.3.2 [Transaction End Indication], page 62).

3.2.1.1 User Primitives Successful Transaction Establishment

The following user primitives support COTS Phase I (Transaction Establishment) services:

- `TR_BEGIN_REQ`: This primitive requests that the TR provider form a transaction association with the specified destination TR user.
- `TR_BEGIN_RES`: This primitive requests that the TR provider accept a previous transaction indication.

3.2.1.2 Provider Primitives Successful Transaction Establishment

The following provider primitives support COTS Phase I (Transaction Establishment) services:

- `TR_BEGIN_IND`: This primitive indicates to the TR user that a transaction association request has been made by a user at the specified source address.
- `TR_BEGIN_CON`: This primitive indicates to the TR user that a transaction initiation request has been confirmed on the specified responding address.

The sequence of primitives in a successful transaction initiation is defined by the time sequence diagrams as shown in Figure 3.6.

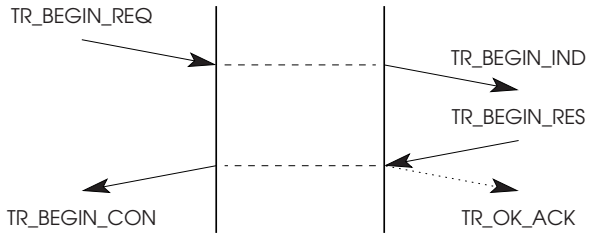

Figure 3.6: *Sequence of Primitives – Successful Transaction Initiation*

The sequence of primitives for the transaction initiation response token value determination is shown in Figure 3.7 (procedures for transaction initiation response token value determination are discussed in Section 4.1.2.1 [Transaction Bind Request], page 26, and Section 4.1.2.2 [Transaction Bind Acknowledgement], page 29).

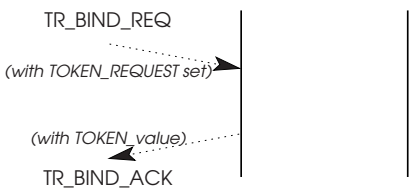

Figure 3.7: *Sequence of Primitives – Transaction Response Token Value Determination*

3.2.2 Transaction Data Transfer Phase

Flow control on the transaction association is done by management of the queue capacity, and by allowing objects of certain types to be inserted to the queues, as shown in *Table 4*.

3.2.2.1 Primitives for Data Transfer

The following primitives support COTS Phase II (Transaction Data Transfer) services:

- `TR_CONT_REQ`: This primitive requests that the TR provider transfer the specified user data.
- `TR_CONT_IND`: This primitive indicates to the TR user that this message contains user data.

Figure 3.8 shows the sequence of primitives for successful user data transfer. The sequence of primitives may remain incomplete if a `TR_END_REQ`, `TR_END_IND`, `TR_ABORT_REQ`, or `TR_ABORT_IND` primitive occurs.

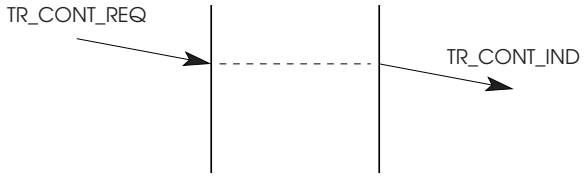

Figure 3.8: *Sequence of Primitives – Data Transfer*

3.2.3 Transaction Termination Phase

The transaction association procedure is initialized by insertion of an end or abort object (associated with a `TR_END_REQ` or `TR_ABORT_REQ`) into the queue. As shown in Table?, the termination procedure is destructive with respect to other objects in the queue, and eventually results in the emptying of queues and termination of the transaction association.

The sequence of primitives depends on the origin of the termination action. The sequence may be:

1. invoked by on TR user, with a request from that TR user leading to an indication to the other;
2. invoked by both TR users, with a request from each of the TR users;
3. invoked by the TR provider, with an indication to each of the TR users;
4. invoked independently by one TR user and the TR provider, with a request from the originating TR user and an indication to the other.

3.2.3.1 Primitives for Transaction Termination

The following primitives support CONS Phase III (Transaction Termination) services:

- `TR_END_REQ`: This primitive requests that the TR provider deny an outstanding request for a transaction association or normal termination of an existing transaction.

- `TR_ABORT_REQ`: This primitive requests that the TR provider deny an outstanding request for a transaction association or abnormal termination of an existing transaction.

- `TR_END_IND`: This primitive indicates to the TR user that either a request for transaction initiation has been denied or an existing transaction has been terminated normally.

- `TR_ABORT_IND`: This primitive indicates to the TR user that either a request for transaction initiation has been denied or an existing transaction has been terminated abnormally.

The sequence of primitives are shown in the time sequence diagrams in the figures that follow:

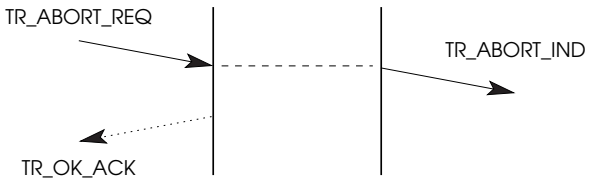

Figure 3.9: *Sequence of Primitives – TR User Invoked Termination*

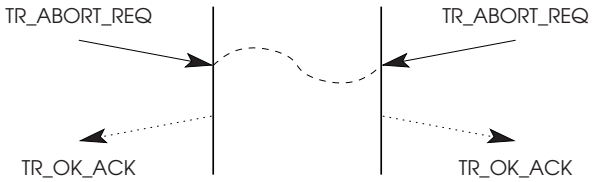

Figure 3.10: *Sequence of Primitives – Simultaneous TR User Invoked Termination*

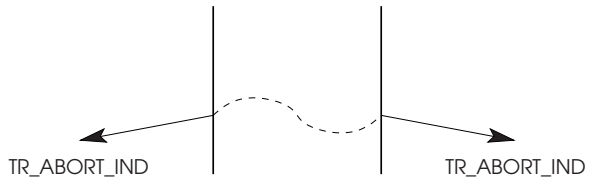

Figure 3.11: *Sequence of Primitives – TR Provider Invoked Termination*

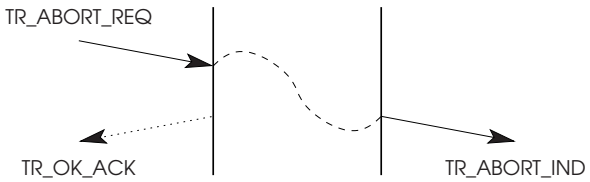

Figure 3.12: *Sequence of Primitives – Simultaneous TR User and Provider Invoked Termination*

A TR user may reject a transaction initiation attempt by issuing a `TR_ABORT_REQ`. The originator parameter in the `TR_ABORT_REQ` will indicate TR user invoked termination. The sequence of primitives is shown in Figure 3.13.

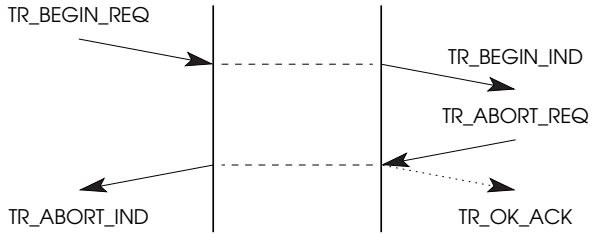

Figure 3.13: *Sequence of Primitives – TR User Rejection of a Transaction Initiation Attempt*

If the TR provider is unable to establish a transaction, it indicates this to the requester by an `TR_ABORT_IND`. The originator of the primitive indicates a TR provider invoked release. This is shown in Figure 3.14.

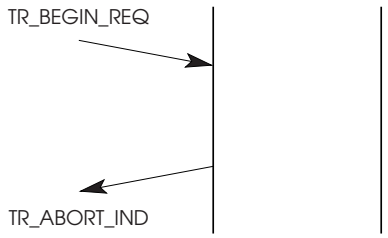

Figure 3.14: *Sequence of Primitives – TR Provider Rejection of a Transaction Initiation Attempt*

3.3 Connectionless Mode Services Definition

The connectionless mode service allows for the transfer of transaction user data in one and both directions simultaneously without establishing a transaction dialogue. A set of primitives are defined that carry transaction user data and control information between the TR user and the TR provider entities. The primitives are modelled as requests initiated by

the TR user and indications initiated by the TR provider. Indications may be initiated by the TR provider independently from requests by the TR user.

The connectionless mode service consists of one phase.

3.3.1 Request and Response Primitives

- `TR_UNI_REQ`: This primitive requests that the TR provider send the transaction user data to the specified destination.
- `TR_UNI_IND`: This primitive indicates to the TR user that a user data sequence has been received from the specified originating address.

Figure 3.15 shows the sequence of primitives for the connectionless mode of transfer.

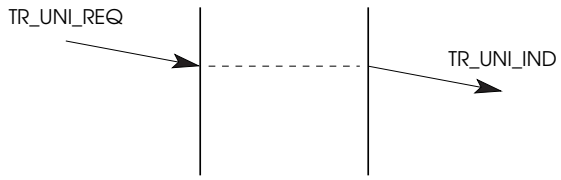

Figure 3.15: *Sequence of Primitives – Connectionless Mode Data Transfer*

- `TR_NOTICE_IND`: This primitive indicates to the TR user that the user data with the specified destination address and QOS parameters produced an error. This primitive is specific to CLTS.

Figure 3.16 shows the sequence of primitives for the CLTS error management primitive.

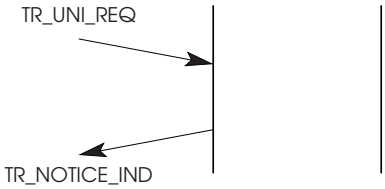

Figure 3.16: *Sequence of Primitives – CLTS Error Indication Service*

4 TRI Primitives

This section describes the format and parameters of the TRI primitives. In addition, it discusses the states in which the primitive is valid, the resulting state, and the acknowledgement that the primitive expects.

The mapping of TRI of TRI primitives to the primitives defined in ITU-T Q.771, ITU-T X.219 and ANSI T1.114 are shown in Appendix A [Mapping TRI Primitives], page 93. The state/event tables for these primitives are shown in Appendix B [State/Event Tables], page 99. The precedence tables for the TRI primitives are shown in Appendix C [Primitive Precedence Tables], page 101.

The following tables provide a summary of the TR primitives and their parameters.

SERVICE	PRIMITIVE	PARAMETERS
TR Initiation	TR_BEGIN_REQ	()
	TR_BEGIN_IND	()
	TR_BEGIN_RES	()
	TR_BEGIN_CON	()

Table 4. *Transaction Initiation Transaction Service Primitives*

SERVICE	PRIMITIVE	PARAMETERS
TR Data Transfer	TR_CONT_REQ	()
	TR_CONT_IND	()

Table 5. *Transaction Data Transfer Transaction Service Primitives*

SERVICE	PRIMITIVE	PARAMETERS
TR Termination	TR_END_REQ	()
	TR_END_IND	()
	TR_ABORT_REQ	()
	TR_ABORT_IND	()

Table 6. *Transaction Termination Transaction Service Primitives*

4.1 Management Primitives

These primitives apply to all transaction modes.

4.1.1 Transaction Information

4.1.1.1 Transaction Information Request

`TR_INFO_REQ`

This primitive request the TR provider to return the values of all supported protocol parameters (see Section 4.1.1.2 [Transaction Information Acknowledgement], page 24), and also the current state of the TR provider (as defined in Appendix B [State/Event Tables], page 99). This primitive does not affect the state of the TR provider 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 TR_info_req {
    ulong PRIM_type;          /* Always TR_INFO_REQ */
} TR_info_req_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type. Always `TR_INFO_REQ`.

Modes

Both connection-mode and connectionless-mode.

Originator

Transaction user.

Valid States

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

New State

The new state remains unchanged.

Rules

For the rules governing the requests made by this primitive, see the `TR_INFO_ACK` primitive described in Section 4.1.1.2 [Transaction Information Acknowledgement], page 24.

Acknowledgements

This primitive requires the TR provider to generate one of the following acknowledgements upon receipt of the primitive and that the TR user wait for the acknowledgement before issuing any other primitives:

— *Successful:* Correct acknowledgement of the primitive is indicated with the `TR_INFO_ACK` primitive described in Section 4.1.1.2 [Transaction Information Acknowledgement], page 24.

— *Non-fatal Errors:* These errors will be indicated with the `TR_ERROR_ACK` primitive described in Section 4.1.4.2 [Transaction Error Acknowledgement], page 42. The allowable errors are as follows:

There are no errors associated with the issuance of this primitive.

4.1.1.2 Transaction Information Acknowledgement

`TR_INFO_ACK`

This primitive indicates to the TR user any relevant protocol-dependent parameters.[1] It should be initiated in response to the `TR_INFO_REQ` primitive described above under Section 4.1.1.1 [Transaction Information Request], page 22.

Format

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

```
typedef struct TR_info_ack {
    long PRIM_type;           /* Always TR_INFO_ACK */
    long ASDU_size;           /* maximum ASDU size */
    long EASDU_size;          /* maximum EASDU size */
    long CDATA_size;          /* connect data size */
    long DDATA_size;          /* discon data size */
    long ADDR_size;           /* address size */
    long OPT_size;            /* options size */
    long TIDU_size;           /* transaction i/f data unit size */
    long SERV_type;           /* service type */
    long CURRENT_state;       /* current state */
    long PROVIDER_flag;       /* type of TR provider */
    long TRI_version;         /* version # of tri that is supported */
} TR_info_ack_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type. Always `TR_INFO_ACK`.

`ASDU_size`
: Indicates the maximum size (in octets) of Transaction Service User Data supported by the TR provider.

`EASDU_size`
: Indicates the maximum size (in octets) of Expedited Transaction Service User Data supported by the TR provider.

`CDATA_size`
: Indicates the maximum number of octets of data that may be associated with a transaction initiation primitive.

`DDATA_size`
: Indicates the maximum number of octets of data that may be associated with a transaction termination primitive.

1

`ADDR_size`
: Indicates the maximum size (in decimal digits) of a protocol address.

`OPT_size`
: Indicates the maximum size (in decimal digits) of the options.

`AIDU_size`
: Indicates the maximum size (in octets) of a Transaction Interface User Data supported by the TR provider. This is the maximum amount of user data octets that can be trasnfered acros the interface in a single data request primitive.

`SERV_type`
: Indicates the service type.

`CURRENT_state`
: Indicates the current interface state.

`PROVIDER_flag`
: Indicates the transaction provider flags.

`TRI_version`
: Indicates the TR version. This is Version 1 of the interface specification.

Modes

This primitive is valid in both connection mode and connectionless mode.

Originator

This primitive is issued by the TR provider.

Valid State

This primitive may be issued in response to a `TR_INFO_REQ` and is valid in any state.

New State

On success, the new state is unchanged; on error, unchanged.

Rules

The following rules apply whey the type is `TR_CLTRS`:

— The `EASDU_size`, `CDATA_size` and `DDATA_size` fields should be '`-2`'.

— The `ASDU_size` should equal the `AIDU_size`.

4.1.2 Transaction Protocol Address Management

4.1.2.1 Transaction Bind Request

`TR_BIND_REQ`

This primitive requests that the TR provider bind a protocol address to the *stream*, negotiate the number of dialogue indications allowed to be outstanding by the TR provider for the specified protocol address, and activate[1] the *stream* associated with the protocol address.

Format

The format of the message is one `M_PROTO` message block. The format of the `M_PROTO` message block is as follows:[2]

```
typedef struct TR_bind_req {
    ulong PRIM_type;          /* Always TR_BIND_REQ */
    ulong ADDR_length;        /* address length */
    ulong ADDR_offset;        /* address offset */
    ulong XACT_number;        /* maximum outstanding transaction reqs. */
    ulong BIND_flags;         /* bind flags */
} TR_bind_req_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Specifies the primitive type. Always `TR_BIND_REQ`.

`ADDR_length`
: Specifies the length[3] of the protocol address to be bound to the *stream*.

`ADDR_offset`
: Specifies the offset from the beginning of the `M_PROTO` message block where the protocol address begins. The proper alignment of the address in the `M_PROTO` message block is not guaranteed. The address in the `M_PROTO` message block is, however, aligned the same as it was received from the TR user.

---

[1] A *stream* is viewed as active when the transaction provider may receive and transmit APDUs (ACSE protocol data units) associated with the *stream*.

[2] The format of the `TR_BIND_REQ` primitive is chosen to be as consistent as possible with the equivalent TPI and NPI primitives.

[3] All lengths, offsets and sizes in all structures refer to the number of octets.

`XACT_number`
: [4]The requested number of dialogue begin indications[5] allowed to be outstanding by the TR provider for the specified protocol address. Only one stream per protocol address is allowed to have a `XACT_number` greater than zero. This indicates to the TR provider that the *stream* is a *listener stream* for the TR user. This *stream* will be used by the TR provider for dialogue "begin" indications for that protocol address, see Section 4.2.1.2 [Transaction Begin Indication], page 48.

`BIND_flags`
: Unused.

Modes

This primitive is valid both in connection and connectionless modes.

Originator

This primitive is issued by the TR user.

Valid State

This primitive is valid in state `TRS_UNBND`.

New State

The new state is `TRS_WACK_BREQ`.

Rules

For the rules governing the requests made by this primitive, see the `TR_BIND_ACK` primitive described in Section 4.1.2.2 [Transaction Bind Acknowledgement], page 29.

Acknowledgements

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

— *Successful:* Correct acknowledgement of the primitive is indicated with the `TR_BIND_ACK` primitive described in Section 4.1.2.2 [Transaction Bind Acknowledgement], page 29.

— *Non-fatal errors:* These errors will be indicated with the `TR_ERROR_ACK` primitive described in Section 4.1.4.2 [Transaction Error Acknowledgement], page 42. The allowable errors are as follows:

`TRBAADDR` Indicates that the protocol address was in an incorrect format or the address contained illegal information. It is not intended to indicate protocol errors.

[4] This field should be ignored by TR providers providing only a unidirectional (TCAP operation class 4, ROSE operation class 5) service.

[5] If the number of outstanding "begin" indications equals `XACT_number`, the TR provider need not discard further incoming "begin" indications, but may choose to queue them internally until the number of outstanding "begin" indications dropts below `XACT_number`.

`TRNOADDR` Indicates that the TR provider could not allocate an address.

`TRACCES` Indicates that the user did not have proper permissions for the use of the requested address.

`TROUTSTATE`
The primitive would place the transaction interface out of state for the indicated transaction.

`TRSYSERR` A system error occurred and the UNIX System error is indicated in the primitive.

`TRADDRBUSY`
Indicates that the requested address is already in use.

4.1.2.2 Transaction Bind Acknowledgement

`TR_BIND_ACK`

This primitive indicates to the TR user that the specified protocol address has been bound to the *stream*, that the specified number of dialogue indications are allowed to be queued by the TR provider for the specified protocol address, and that the *stream* associated with the specified protocol address has been activated.

Format

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

```
typedef struct TR_bind_ack {
    ulong PRIM_type;          /* Always TR_BIND_ACK */
    ulong ADDR_length;        /* address length */
    ulong ADDR_offset;        /* address offset */
    ulong XACT_number;        /* open transactions */
    ulong TOKEN_value;        /* value of "token" assigned to stream */
} TR_bind_ack_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type. Always `TR_BIND_ACK`.

`ADDR_length`
: Indicates the length of the protocol address that was bound to the *stream*.

`ADDR_offset`
: Indicates the offset from the beginning of the `M_PCPROTO` message block where the protocol address begins. The proper alignment of the address in the `M_PCPROTO` message block is not guaranteed.

`XACT_number`
: [1] Indicates the accepted number of dialogue indications allowed to be outstanding by the TR provider for the specified protocol address.

`TOKEN_value`
: Indicates a token value to be used when accepting dialogues indicated on other *streams* using this *stream*.

Modes

This primitive is valid in bidirectional and unidirectional modes.

Originator

This primitive is issued by the TR provider.

[1] This field does not apply to unidirectional TR providers.

Valid State

This primitive is issued in response to a TR_BIND_REQ and is valid in state TRS_WACK_BREQ.

New State

On success, the new state is TRS_IDLE; on error, TRS_UNBND.

Rules

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

— If the ADDR_length field in the TR_BIND_REQ primitive is zero (0), then the TR provider must assign a protocol address to the user.

— The TR provider is to bind the protocol address as specified in the TR_BIND_REQ primitive. If the requested protocol address is in use or if the TR provider cannot bind the specified address, it must return an error.

The following rules apply to negotiating the XACT_number argument:

— The returned value must be less than or equal to the corresponding requested number as indicated in the TR_BIND_REQ primitive.

— If the requested value is greater than zero, the returned value must also be greater than zero.

— Only one *stream* that is bound to the indicated protocol address any have a negotiated accepted number of maximum transaction requests greater than zero. If a TR_BIND_REQ primitive specifies a value greater than zero, but another *stream* has already bound itself to the given protocol address with a value greater than zero, the TR provider must return an error.

— If a *stream* with XACT_number greater than zero is used to accept a dialogue (without specifying a TRANS_id), the *stream* will be found busy during the duration of that connection and no other *streams* may be bound to that protocol address with a XACT_number greater than zero. This will prevent more than one *stream* bound to the identical protocol address from accepting dialogue indications. See also Section 4.2.1.3 [Transaction Begin Response], page 50.

— A *stream* requesting a XACT_number of zero should always be legal. This indicates to the TR provider that the *stream* is to be used to request dialogues only.

— *stream* with a negotiated XACT_number greater than zero may generate dialogue requests (see Section 4.2.1.1 [Transaction Begin Request], page 45,) or accept dialogue indications (see Section 4.2.1.3 [Transaction Begin Response], page 50.)

If the above rules result in an error condition, then the TR provider must issue a TR_ERROR_ACK primitive to the TR user specifying the error as defined in the description of the TR_BIND_REQ primitive, Section 4.1.2.1 [Transaction Bind Request], page 26.

4.1.2.3 Transaction Unbind Request

`TR_UNBIND_REQ`

This primitive requests that the TR provider unbind the protocol address previously associated with the *stream* and deactivate the *stream*.

Format

The format of the message is one `M_PROTO` message block structured as follows:

```
typedef struct TR_unbind_req {
    ulong PRIM_type;          /* Always TR_UNBIND_REQ */
} TR_unbind_req_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Specifies the primitive type. Always `TR_UNBIND_REQ`.

Mode

This primitive is valid in both unidirectional and bidirectional modes.

Originator

This primitive is originated by the TR user.

Valid State

This primitive is valid in state `TRS_IDLE`.

New State

The new state is `TRS_WACK_UREQ`.

Acknowledgements

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

— *Successful:* Correct acknowledgement of the primitive is indicated with the `TR_OK_ACK` primitive described in Section 4.1.4.1 [Transaction Successful Receipt Acknowledgement], page 41.

— *Non-fatal errors:* These errors will be indicated with the `TR_ERROR_ACK` primitive described in Section 4.1.4.2 [Transaction Error Acknowledgement], page 42. The allowable errors are as follows:

`TROUTSTATE`
: The primitive would place the transaction interface out of state for the indicated transaction.

`TRSYSERR`
: A system error occurred and the UNIX System error is indicated in the primitive.

4.1.2.4 Transaction Protocol Address Request

`TR_ADDR_REQ`

This primitive requests that the TR provider return the local protocol address that is bound to the *stream* and the address of the remote ASE if a transaction association has been established.

Format

The format of the message is one `M_PROTO` message block structured as follows:

```
typedef struct TR_addr_req {
    long PRIM_type;           /* always TR_ADDR_REQ */
    ulong TRANS_id;           /* Transaction id */
} TR_addr_req_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Specifies the primitive type. Always `TR_ADDR_REQ`.

`TRANS_id`
: Specifies the transaction association identifier for which address service is requested. If address service is requested for local bind address only, then the transaction identifier must be '`-1`'.

Mode

This primitive is valid in both unidirectional and bidirectional modes.

Originator

This primitive is originated by the TR user.

Valid State

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

New State

The new state is unchanged.

Rules

For the rules governing the requests made by this primitive, see the `TR_ADDR_ACK` primitive described in Section 4.1.2.5 [Transaction Protocol Address Acknowledgement], page 34.

Acknowledgements

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

— *Successful:* Correct acknowledgement of the primitive is indicated with the `TR_ADDR_ACK` primitive described in Section 4.1.2.5 [Transaction Protocol Address Acknowledgement], page 34.

— *Non-fatal errors:* These errors will be indicated with the `TR_ERROR_ACK` primitive described in Section 4.1.4.2 [Transaction Error Acknowledgement], page 42. The allowable errors are as follows:

`TRBADID` The transaction identifier specified in the primitive was incorrect or invalid.

`TRNOTSUPPORT`
This primitive is not supported by the transaction provider.

`TRSYSERR` A system error has occured and the Linux system error is indicated in the primitive.

4.1.2.5 Transaction Protocol Address Acknowledgement

`TR_ADDR_ACK`

This primitive indicates to the TR user the addresses of the local and remote ASE. The local address is the protocol address that has been bound to the *stream*. If an transaction association has been established, the remote address is the protocol address of the remote ASE.

Format

The format of the message is one `M_PCPROTO` message block structured as follows:

```
typedef struct TR_addr_ack {
    long PRIM_type;           /* always TR_ADDR_ACK */
    long LOCADDR_length;      /* length of local address */
    long LOCADDR_offset;      /* offset of local address */
    long REMADDR_length;      /* length of remote address */
    long REMADDR_offset;      /* offset of remote address */
} TR_addr_ack_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type. Always `TR_ADDR_ACK`.

`LOCADDR_length`
: Indicates the length of the protocol address that was bound to the *stream*.

`LOCADDR_offset`
: Indicates the offset from the beginning of the `M_PCPROTO` message block where the protocol address begins.

`REMADDR_length`
: Indicates the length of the protocol address of the remote ASE.

`REMADDR_offset`
: Indicates the offset from the beginning of the `M_PCPROTO` message block where the protocol address begins.

The proper alignement of the addresses in the `M_PCPROTO` message block is not guaranteed.

Modes

Both connection-mode and connectionless-mode.

Originator

Transaction provider.

Valid State

This primitive is issued in response to a TR_ADDR_REQ primitive and is valid in any state where a response is pending to a TR_ADDR_REQ.

New State

The new state remains unchanged.

Rules

The following rules apply:

— If the requested transaction identifier was '-1' in the corresponding TR_ADDR_REQ primitive, and the transaction endpoint is not bound to a local address, (i.e. it is in the TRS_UNINIT or TRS_UNBND state) the LOCADDR_length and LOCADDR_offset fields must be set to '0'.

— If the requested transaction exists as identifed in the corresponding TR_ADDR_REQ primitive, LOCADDR_length and LOCADDR_offset fields will be populated to reflect the local association address for the specified transaction.

— If the requested transaction identifier was '-1' in the corresponding TR_ADDR_REQ primitive, the REMADDR_length and REMADDR_offset fields must be set to '0'.

— If the requested transaction exists as identified in the corresponding TR_ADDR_REQ primitive, REMADDR_length and REMADDR_offset fields will be populated to reflect the remote association address for the specified transaction.

4.1.3 Transaction Options Management

4.1.3.1 Transaction Options Management Request

`TR_OPTMGMT_REQ`

This primitive alllows the transaction user to manage the options associated with the *stream*. The format of the message is one `M_PROTO` message block.

Format

The format of the message is one `M_PCPROTO` message block structured as follows:

```
typedef struct TR_optmgmt_req {
    ulong PRIM_type;          /* Always TR_OPTMGMT_REQ */
    ulong OPT_length;         /* options length */
    ulong OPT_offset;         /* options offset */
    ulong MGMT_flags;         /* options data flags */
} TR_optmgmt_req_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Specifies the primitive type. Always `TR_OPTMGMT_REQ`.

`OPT_length`
: Specifies the length of the protocol options associated with the primitive.

`OPT_offset`
: Specifies the offset from the beginning of the `M_PROTO` message block where the options begin.

`MGMT_flags`
: Specifies the management flags which define the request made by the transaction user.

The proper alignment of the options is not guaranteed. The options are, however, aligned the same as received from the transaction user.

Flags

The allowable flags are:

`TR_NEGOTIATE`
: Negotiate and set the options with the transaction provider.

`TR_CHECK` Check the validity of the specified options.

`TR_DEFAULT`
: Return the default options.

`TR_CURRENT`
: Return the currently effective option values.

Modes

This primitive is valid both in unidirectional and bidirectional modes.

Originator

This primitive is originated by the transaction user.

Valid State

This primitive is valid in any state where the transaction user is not expecting a local acknowledgement.

New State

The state remains unchanged.

Rules

For the rules governing the requests made by this primitive, see the `TR_OPTMGMT_ACK` primitive described in Section 4.1.3.2 [Transaction Options Management Acknowledgement], page 38.

Acknowledgements

This primitive requires the TR provider to generate one of the following acknowledgements upon receipt of the primitive, and that the transaction user wait for the acknowledgement before issuing any other primitives:

— *Successful:* Correct acknowledgement is indicated with the `TR_OPTMGMT_ACK` primitive described in Section 4.1.3.2 [Transaction Options Management Acknowledgement], page 38.

— *Non-fatal errors:* These errors will be indicated with the `TR_ERROR_ACK` primitive described in Section 4.1.4.2 [Transaction Error Acknowledgement], page 42. The allowable errors are as follows:

`TRACCES` The user did not have proper permissions for the use of the requested options.

`TRBADFLAG`
The flags as sepcified were incorrect or invalid.

`TRBADOPT` The options as specified were in an incorrect ofrmat, or they contained invalid information.

`TROUTSTATE`
The primitive would place the transaction interface out of state for the indicated transaction.

`TRNOTSUPPORT`
This primiitve is not supported by the transaction provider.

`TRSYSERR` A system error occurred and the UNIX System error is indicated in the primitive.

4.1.3.2 Transaction Options Management Acknowledgement

`TR_OPTMGMT_ACK`

This primitive indicates to the transaction user that the options management request has completed.

Format

The format of the message is one `M_PCPROTO` message block structured as follows:

```
typedef struct TR_optmgmt_ack {
    ulong PRIM_type;          /* Always TR_OPTMGMT_ACK */
    ulong OPT_length;         /* options length */
    ulong OPT_offset;         /* options offset */
    ulong MGMT_flags;         /* options data flags */
} TR_optmgmt_ack_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type. Always `TR_OPTMGMT_ACK`.

`OPT_length`
: Indicates the length of the protocol options associated with the primitive.

`OPT_offset`
: Indicates the offset from the beginning of the `M_PCPROTO` message block where the options begin. The proper alignment of the options is not guaranteed.

`MGMT_flags`
: Indicates the management flags in the same form as specified in the `TR_OPTMGMT_REQ` primitive, See Section 4.1.3.1 [Transaction Options Management Request], page 36, with any additional flags as specified below.

Flags

The flags returned in `MGMT_flags` represents the single most severe result of the operation. The flags returned will be one of the following values (in order of decreasing severity):

`TR_NOTSUPPORT`
: This flag indicates that at least one of the options specified in the `TR_OPTMGMT_REQ` primitive was not supported by the trasnaction provider at the current privilege level of the requesting user.

`TR_READONLY`
: This flag indicates that at least one of the options specified in the `TR_OPTMGMT_REQ` primitive is read-only (for the current TRI state). This flag does not apply when the `MGMT_flags` field in the `TR_OPTMGMT_REQ` primitive was `T_DEFAULT`.

`TR_FAILURE`
: This flag indicates that negotiation of at least one of the options specified in the `TR_OPTMGMT_REQ` primitive failed. This is not used for illegal format or values. This flag does not apply when the `MGMT_flags` field in the `TR_OPTMGMT_REQ` primitive was `T_DEFAULT` or `T_CURRENT`.

`TR_PARTSUCCESS`
: This flag indicates that the negotiation of at least one of the options specified in the `TR_OPTMGMT_REQ` primitive was negotiated to a value of lesser quality than the value requested. This flag only applies when the `MGMT_flags` field of the `TR_OPGMGMT_REQ` primitive was `T_NEGOTIATE`.

`TR_SUCCESS`
: This flag indicates that all of the specified options were negoitated or returned successfully.

Mode

This primitive is valid in both unidirectional and bidirectional modes.

Originator

This primitive is originated by the TR provider.

Valid State

This primitive is issued in response to a `TR_OPTMGMT_REQ` primitive and is valid in any state.

New State

The new state remains unchanged.

Rules

The following rules apply to the `TR_OPTMGMT_ACK` primitive:

— If the value of `MGMT_flags` in the `TR_OPTMGMT_REQ` primitive is `TR_DEFAULT`, the provider should return the default provider options without changing the existing options associated with the *Stream*.

— If the value of `MGMT_flags` in the `TR_OPTMGMT_REQ` primitive is `TR_CHECK`, the provider should return the options as specified in the `TR_OPTMGMT_REQ` primitive along with the additional flags `TR_SUCCESS` or `TR_FAILURE` which indicate to the user whether the specified options are supportable by the provider. The provider should not change any existing options associated with the *Stream*.

— If the value of `MGMT_flags` in the `TR_OPTMGMT_REQ` primitive is `TR_NEGOTIATE`, the provider should set and negotiate the option as specified by the following rules:

  – If the `OPT_length` field of the `TR_OPTMGMT_REQ` primitive is zero (‘`0`’), then the transaction provider is to set and return the default options associated with the *Stream* in the `TR_OPTMGMT_ACK` primitive.

  – If options are specified in the `TR_OPTMGMT_REQ` primitive, then the transaction provider should negotiate those options, set the negotiated options and return the

negotiated options in the TR_OPTMGMT_ACK pirmitive. It is the user's responsibility to check the negotiated options returned in the TR_OPMGMT_ACK primitive and take appropriate action.

— If the value of MGMT_flags in the TR_OPTMGMT_REQ primtiive is TR_CURRENT, the provider should return the currently effective option values without changing any existing options associated with the *Stream*.

Errors

If the above rules result in an error condition, the transaction provider must issue a TR_ERROR_ACK primitive (see Section 4.1.4.2 [Transaction Error Acknowledgement], page 42) to the transaction user specifying the error as defined in the description of the TR_OPTMGMT_REQ primitive (see Section 4.1.3.1 [Transaction Options Management Request], page 36).

4.1.4 Transaction Error Management

4.1.4.1 Transaction Successful Receipt Acknowledgement

`TR_OK_ACK`

This primitive indicates to the TR user that the previous TR-user-originated primitive was received successfully by the TR provider. It does not indicate to the TR user any TR protocol action taken due to the issuance of the last primitive. This may only be initiated as an acknowledgement for those primitives that require one.

Format

The format of the message is one `M_PCPROTO` message block structured as follows:

```
typedef struct TR_ok_ack {
    ulong PRIM_type;          /* Always TR_OK_ACK */
    ulong CORRECT_prim;       /* correct primitive */
} TR_ok_ack_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type. Always `TR_OK_ACK`.

`CORRECT_prim`
: Indicates the primitive type that was successfully received.

Modes

This primitive is valid in all Operations Classes.

Originator

This primitive is issued by the TR provider.

Valid State

Valid in any state where a local acknowledgement requiring `TR_OK_ACK` response is pending.

New State

Depends on the current state; see Appendix B [State/Event Tables], page 99.

4.1.4.2 Transaction Error Acknowledgement

`TR_ERROR_ACK`

This primitive indicates to the TR user that a non-fatal[1] error has occurred in the last TR-user-originated primitive. This may only be initiated as an acknowledgement for those primitives that require one. It also indicates to the TR user that no action was taken on the primitive that cause the error.

Format

The format of the message is one `M_PCPROTO` message block structured as follows:

```
typedef struct TR_error_ack {
    ulong PRIM_type;          /* Always TR_ERROR_ACK */
    ulong ERROR_prim;         /* primitive in error */
    ulong TRI_error;          /* TRI error code */
    ulong UNIX_error;         /* UNIX error code */
    ulong TRANS_id;           /* Transaction id */
} TR_error_ack_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type. Always `TR_ERROR_ACK`.

`ERROR_prim`
: Indicates the primitive type that was in error.

`TRI_error`
: Indicates the Transaction Sub-Layer Interface error code.

`UNIX_error`
: Indicates the UNIX System error code. This field is zero (0) unless the `TRI_error` is equal to `TRSYSERR`.

`TRANS_id`
: Indicaets the transcation identifier for the transaction upon which the primitive caused an error.

Mode

This primitive can be issued in any Operations Class.

Originator

This primitive is originated by the TR provider.

[1] For an overview of the error handling capabilities available to the TR provider, see Chapter 5 [Diagnostics Requirements], page 73.

Valid State

This primitive is valid in any state where a local acknowledgement is pending and an error has occurred.

New State

The new state is the state that the interface was in before the primitive in error was issued, see Appendix B [State/Event Tables], page 99.

Rules

This primitive may only be issued as an acknowledgement for those primitives that require one. It also indicates to the user that no action was taken on the primtiive that caused the error.

Errors

The TR provider is allowed to return any of the following TR error codes:

`TRBADADDR`
: Indicates that the protocol address as specified in the primitive was of an incorrect format or the address contained illegal information.

`TRBADOPT` Indicates that the options as specified in the primitive were in an incorrect format, or they contained illegal information.

`TRBADF` Indicates that the *stream* queue pointer as specified in the primitive was illegal.

`TRNOADDR` Indicates that the TR provider could not allocate a protocol address.

`TRACCES` Indicates that the user did not have proper permissions to use the protocol address or options specified in the primitive.

`TROUTSTATE`
: Indicates that the primitive would place the interface out of state.

`TRBADSEQ` Indicates that the transaction identifier specified in the primitive was incorrect or illegal.

`TRBADFLAG`
: Indicates that the flags specified in the primitive were incorrect or illegal.

`TRBADDATA`
: Indicates that the amount of user data specified was illegal.

`TRSYSERR` Indicates that a system error has occurred and that the UNIX System error is indicated in the primitive.

`TRADDRBUSY`
: Indicates that the requested address is already in use.

`TRRESADDR`
: Indicates that the TR provider requires the responding *stream* be bound to the same protocol address as the *stream* on which the dialogue "begin" indication (see Section 4.2.1.2 [Transaction Begin Indication], page 48) was received.

`TRNOTSUPPORT`
: Indicates that the TR provider does not support the requested capability.

4.2 Connection-Oriented Mode Primitives

4.2.1 Transaction Establishment

The transaction begin service provides means to start a transaction between two TR-users. This may be accompanied by the transfer of TR-user information contained in `M_DATA` message blocks accompanying the primitive.

4.2.1.1 Transaction Begin Request

`TR_BEGIN_REQ`

This primitive requests that the TR provider form an transaction association to the specified destination protocol address.

Format

The format of the message is one `M_PROTO` message block, followed by zero or more `M_DATA` message blocks if any user data is specified by the TR user. The format of the `M_PROTO` message block is as follows:

```
typedef struct TR_begin_req {
    ulong PRIM_type;          /* Always TR_BEGIN_REQ */
    ulong CORR_id;            /* Correlation Id */
    ulong ASSOC_flags;        /* Association flags */
    ulong DEST_length;        /* Destination address length */
    ulong DEST_offset;        /* Destination address offset */
    ulong ORIG_length;        /* Originating address length */
    ulong ORIG_offset;        /* Originating address offset */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_begin_req_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Specifies the primitive type: always `TR_BEGIN_REQ`.

`CORR_id`
: Specifies the correlation identifier for the newly formed transaction. The correlation identifier is an identifier chose by the TR user that uniquely identifies this transaction association establishment request from other establishment requests on the same *stream*. If the `CORR_id` is zero (0), it specifies that this is the only transaction to be formed on the requesting *stream* and attempts to form additional transactions before this transaction is complete will fail. The value of `CORR_id` will be returned in

`ASSOC_flags`
: Specifies the option flags provided with the primitive. See “Flags” below. Some flags may be provider specific.

`DEST_length`
: Specifies the length of the protocol address to which to establish an transaction association.

`DEST_offset`
: Specifies the offset from the beginning of the `M_PROTO` message block where the protocol address begins.

`ORIG_length`
: Specifies the length of the protocol address from which to establish an transaction association.

`ORIG_offset`
: Specifies the offset from the beginning of the `M_PROTO` message block where the protocol address begins.

`OPT_length`
: Specifies the length of the protocol options associated with the transaction.

`OPT_offset`
: Specifies the offset from the beginning of the `M_PROTO` message block where the protocol options begin.

Flags

`TR_SEQ_ASSURANCE`
: By setting this flag on the primitive, the originating transaction user can indicate that "sequence assured" service is requested from the underlying network service provider.

`TR_NO_PERMISSION`
: By setting this flag on the primitive, the originating transaction user can either deny (set) or grant (clear) permission for the transaction peer to terminate the transaction association upon receipt of the corresponding primitive at the peer (see Section 4.2.1.2 [Transaction Begin Indication], page 48). This flag can only be used with transaction provider that support it (see [Addendum for ANSI Conformance], page 85).

Valid State

This primitive is valid in transaction state `TRS_IDLE`. This primitive is only valid in connection-oriented mode.

New State

The new state for the interface is `TRS_WACK_CREQ`.

Rules

The following rules apply to the specification of parameters to this primitive:

— When the originating address is not specified, `ORIG_length` and `ORIG_offset` must be specified as zero (0).

— When the ORIG_length and ORIG_offset are zero (0), the originating address is the local address that is implicitly associated with the access point from the local bind service (see Section 4.1.2.1 [Transaction Bind Request], page 26).

— The destination address must be specified and the TR provider will return error TRNOADDR if the DEST_length and DEST_offset are zero (0).

Acknowledgements

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

— *Successful Association Establishment:* This is indicated with the TR_BEGIN_CON primitive described in Section 4.2.1.1 [Transaction Begin Request], page 45. This results in the TRS_DATA_XFER state for the transaction. Successful establishment and tear down can also be indicated with the TR_END_IND primitive described in Section 4.2.3.2 [Transaction End Indication], page 62. This results in the TRS_IDLE state for the transaction.

— *Unsuccessful Association Establishment:* This is indicated with the TR_ABORT_IND primitive described in Section 4.2.3.4 [Transaction Abort Indication], page 66. For example, an association may be rejected because either the called transaction user cannot be reached, or the transaction provider or the called transaction user did not agree on the specified options. This results in the TRS_IDLE state for the transaction.

— *Non-fatal errors:* These are indicated with the TR_ERROR_ACK primitive. The applicable non-fatal errors are defined as follows:

TRACCES This indicates that the user did not have proper permissions for the use of the requested protocol address or protocol options.

TRBADADDR
This indicates that the protocol address was in an incorrect format or the address contained illegal information. It is not intended to indicate protocol connection errors, such as an unreachable destination. Those types of errors are indicated with the TR_ABORT_IND primitive described in Section 4.2.3.4 [Transaction Abort Indication], page 66.

TRBADOPT This indicates that the options were in an incorrect format or they contained illegal information.

TROUTSTATE
The primitive would place the transaction interface out of state.

TRBADDATA
The amount of user data specified was illegal (see Section 4.1.1.2 [Transaction Information Acknowledgement], page 24).

TRSYSERR A system error has occured and the UNIX System error is indicated in the primitive.

4.2.1.2 Transaction Begin Indication

`TR_BEGIN_IND`

This primitive indicates to the destination TR user that a transaction association begin request has been made by the user at the specified source protocol address.

Format

The format of the message is one `M_PROTO` message block, followed by zero or more `M_DATA` message blocks containing user data for the association, structured as follows:

```
typedef struct TR_begin_ind {
    ulong PRIM_type;          /* Always TR_BEGIN_IND */
    ulong TRANS_id;           /* Transaction id */
    ulong ASSOC_flags;        /* Association flags */
    ulong DEST_length;        /* Destination address length */
    ulong DEST_offset;        /* Destination address offset */
    ulong ORIG_length;        /* Originating address length */
    ulong ORIG_offset;        /* Originating address offset */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_begin_ind_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type: always `TR_BEGIN_IND`.

`TRANS_id`
: Indicates the transaction identifier associated by the transaction provider with this begin indication.

`ASSOC_flags`
: Specifies the option flags provided with the primitive. See “Flags” below. Some flags may be provider specific.

`DEST_length`
: Indicates the length of the protocol address to which a transaction association was requested established by the peer.

`DEST_offset`
: Indicates the offset from the beginning of the `M_PROTO` message block where the protocol address begins.

`ORIG_length`
: Indicates the length of the protocol address from which a transaction association was requested established.

`ORIG_offset`
: Indicates the offset from the beginning of the `M_PROTO` message block where the protocol address begins.

`OPT_length`
: Indicates the length of the protocol options associated with the transaction begin indication.

`OPT_offset`
: Indicates the offset from the beginning of the `M_PROTO` message block where the protocol options begin.

Flags

`TR_NO_PERMISSION`
: The value of this flag may indicate either that the transaction peer gives permission (clear) to end the transaction association or refuses permission (set) to end the transaction association. This flag is only valid for transaction providers that support it (see [Addendum for ANSI Conformance], page 85).

Valid State

This primitive is valid in state `TRS_IDLE`. This primitive is only valid in connection-oriented mode.

New State

The new state for the identified transaction is `TRS_WRES_CIND`.

Rules

The following rules apply to the issuance of this primitive by the transaction provider:

— The transaction identifier provided by the transaction provider uniquely identifies this transaction begin indication within the stream upon which the primitive is issued. This must be a positive, non-zero value. The high bit of the transaction identifier is reserved for exclusive use by the transaction user in generating correlation identifiers.

— It is not necessary to indicate a destination address in `DEST_length`, and `DEST_offset` when the protocol address to which the begin indication corresponds is the same as the local protocol address to which the listening stream is bound. In the case that the destination protocol address is not provided, `DEST_length` and `DEST_offset` must both be set to zero (0). When the local protocol address to which the begin indication corresponds is not the same as the bound address for the stream, the transaction provider must indicate the destination protocol address using `DEST_length` and `DEST_offset`.

— The origination protocol address is a mandatory field. The transaction provider must indicate the originating protocol address corresponding to the begin indication using the `ORIG_length` and `ORIG_offset` fields.

— Any indicated options are included in the `OPT_length` and `OPT_offset` fields.

— When the `TR_NO_PERMISSION` flag is set, the transaction user must not issue a `TR_END_REQ` primitive in response to this indication.

4.2.1.3 Transaction Begin Response

TR_BEGIN_RES

This primitive allows the destination TR user to request that the transaction provider accept a previous transaction association begin indication.

Format

The format of the message is one M_PROTO message block, followed by zero or more M_DATA message blocks containing user data for the association, structured as follows:

```
typedef struct TR_begin_res {
    ulong PRIM_type;          /* Always TR_BEGIN_RES */
    ulong TRANS_id;           /* Transaction id */
    ulong ASSOC_flags;        /* Association flags */
    ulong ORIG_length;        /* Originating address length */
    ulong ORIG_offset;        /* Originating address offset */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_begin_res_t;
```

Parameters

The primitive has the following arguments:

PRIM_type
: Specifies the primitive type: always TR_BEGIN_RES.

TRANS_id
: Specifies the transaction identifier of an outstanding begin indication to which the transaction user is responding.

ASSOC_flags
: Specifies the option flags provided with the primitive. See "Flags" below. Some flags may be provider specific.

ORIG_length
: Specifies the length of the protocol address to be used as the responding address.

ORIG_offset
: Specifies the offset from the beginning of the M_PROTO message block where the protocol address begins.

OPT_length
: Specifies the length of the protocol options to be associated with the begin response.

OPT_offset
: Specifies the offset from the beginning of the M_PROTO message block where the protocol options begin.

Flags

`TR_SEQ_ASSURANCE`
: By setting this flag on the primitive, the originating transaction user can indicate that "sequence assured" service is requested from the underlying network service provider.

`TR_NO_PERMISSION`
: By setting this flag on the primitive, the originating transaction user can either deny (set) or grant (clear) permission for the transaction peer to terminate the transaction association upon receipt of the corresponding primitive at the peer (see Section 4.2.1.2 [Transaction Begin Indication], page 48). This flag can only be used with transaction provider that support it (see [Addendum for ANSI Conformance], page 85).

Valid State

This primitive is valid in transaction state `TRS_WRES_CIND`. This primitive is only valid in connection-oriented mode.

New State

The new state for the specified transaction is `TRS_DATA_XFER`.

Rules

Acknowledgements

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

— *Successful:* Correct acknowledgement of the primitive is indicated with the `TR_OK_ACK` primitive described in Section 4.1.4.1 [Transaction Successful Receipt Acknowledgement], page 41.

— *Unsuccessful (Non-fatal errors):* These errors will be indicated with the `TR_ERROR_ACK` primitive described in Section 4.1.4.2 [Transaction Error Acknowledgement], page 42. The allowable errors are as follows:

`TRBADF` The token specified is not associated with an open stream.

`TRBADOPT` The options were in an incorrect format, or they contained illegal information.

`TRACCES` The user did not have proper permissions for the use of the responding protocol address or protocol options.

`TROUTSTATE`
: The primitive would place the transaction interface out of state for the indicated transaction.

`TRBADDATA`
: The amount of user data specified was outside the range supported by the transaction provider.

`TRBADSEQ` The transaction identifier specified in the primitive was incorrect or illegal.

`TRSYSERR` A system error occurred and the UNIX System error is indicated in the primitive.

`TRRESADDR`
The transaction provider requires that the responding *stream* is bound to the same address as the *stream* on which the transaction association begin indication was received.

`TRBADADDR`
This indicates that the protocol address was in an incorrect format or the protocol address contained illegal information.

4.2.1.4 Transaction Begin Confirmation

`TR_BEGIN_CON`

This primitive indicates to the source transaction user that a previous transaction association begin request has been confirmed on the specified responding protocol address.

Format

The format of the message is one `M_PROTO` message block, followed by zero or more `M_DATA` message blocks containing user data for the association, structured as follows:

```
typedef struct TR_begin_con {
    ulong PRIM_type;          /* Always TR_BEGIN_CON */
    ulong CORR_id;            /* Correlation Id */
    ulong TRANS_id;           /* Transaction id */
    ulong ASSOC_flags;        /* Association flags */
    ulong ORIG_length;        /* Originating address length */
    ulong ORIG_offset;        /* Originating address offset */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_begin_con_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type: always `TR_BEGIN_CON`.

`CORR_id`
: Indicates the correlation identifier used by the transport user to uniquely identify the transaction begin request of the stream to which this confirmation corresponds. This is the transaction user assigned transaction identifier of the corresponding `TR_BEGIN_REQ` that this message is confirming.

`TRANS_id`
: Indicates the transaction identifier provided by the transport provider to uniquely identify the transaction on this stream.

`ASSOC_flags`
: Indicates the option flags provided with the primitive. See “Flags” below. Some flags may be provider specific.

`ORIG_length`
: Indicates the length of the responding protocol address from which the confirmation was received.

`ORIG_offset`
: Indicates the offset from the beginning of the `M_PROTO` message block where the responding protocol address begins.

`OPT_length`
: Indicates the length of the confirmed protocol options negotiated by the transaction peer.

`OPT_offset`
: Indicates the offset from the beginning of the `M_PROTO` message block where the confirmed protocol options begin.

The proper alignment of the responding address and options in the `M_PROTO` message block is not guaranteed.

Flags

The following association flags are defined:

`TR_NO_PERMISSION`
: The value of this flag may indicate either that the transaction peer gives permission (clear) to end the transaction association or refuses permission (set) to end the transaction association. This flag is only valid for transaction providers that support it (see [Addendum for ANSI Conformance], page 85).

Mode

This primitive is only valid in connection-oriented mode.

Originator

Transaction provider.

Valid State

This primitive is valid in transaction state `TRS_WCON_CREQ`.

New State

The new state for the transaction is `TRS_DATA_XFER`.

Rules

The following rules apply to the issuance of this primitive:

— It is not always necessary for the transport provider to provide the responding address in the `ORIG_length` and `ORIG_offiset` fields. Where the responding protocol address is the same as the destination protocol address for which the transaction initialization was requested, it is not necessary to provide the responding address in the *TR_BEGIN_CON*. Where the responding protocol address is not provided, the `ORIG_length` and `ORIG_offset` fields are set to zero (0).

— When the `TR_NO_PERMISSION` flag is set, the transaction user must not issue a `TR_END_REQ` primitive in response to this indication.

4.2.2 Transaction Data Transfer

The data transfer service primitives provide for an exchange of transaction user data known as TSDUs, in either direction or in both directions simultaneously on a transaction association. The transaction service preserves both the sequence and the boundaries of the TSDUs.

4.2.2.1 Transaction Continue Request

`TR_CONT_REQ`

This user-originated primitive specifies to the transaction provider that this message contains transaction user data. It allows the transfer of transaction user data between transaction users, without modification by the transaction provider.

The transaction user must send an integral number of octets of data greater than zero. In a case where the size of the TSDU exceeds the TIDU (as specified by the size of the `TIDU_size` parameter of the `TR_INFO_ACK` primitive described in Section 4.1.1.2 [Transaction Information Acknowledgement], page 24), the TSDU may be broken up into more than one TIDU. When a TSDU is broken up into more than one TIDU, the `T_MORE` flag will be set on each TIDU except the last one.

Format

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

a. to indicate that the TSDU is broken into more than one TIDU, and that the data carried in the following `M_DATA` message block constitutes one TIDU;

b. to indicate whether receipt confirmation is desired for the TSDU.

message block, followed by zero or more `M_DATA` message blocks containing user data for the association, structured as follows:

```
typedef struct TR_cont_req {
    ulong PRIM_type;          /* Always TR_CONT_REQ */
    ulong TRANS_id;           /* Transaction id */
    ulong ASSOC_flags;        /* Association flags */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_cont_req_t;
```

Guidelines for use of `M_PROTO`

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

1. The `M_PROTO` message block need not be present when the TSDU size is less that or equal to the TIDU size and one of the following is true:
   - — receipt confirmation has been negotiated for non-use; or
   - — receipt confirmation has been successfully negotiated for use or non-use and the default selection as specified via the `TR_OPTMGMT_REQ` primitive is to be used.

2. The M_PROTO message block must be present when:
   — the TSDU size is greater than the TIDU size;
   — receipt confirmation has been successfully negotiated for use and the default selection as specified with the TR_OPTMGMT_REQ primitive needs to be overridden.

Parameters

The primitive has the following arguments:

PRIM_type
: Specifies the primitive type: always TR_CONT_REQ.

TRANS_id
: Specifies the transaction identifier previously indicated by the transport provider to uniquely identify the transaction. The transaction identifier must be specified by the transaction user unless there is only one transaction supported by the stream in transaction state TRS_DATA_XFER. When specified, the transaction identifier must be the same as the transaction identifier that was indicated by the transaction provider in the corresponding TR_BEGIN_IND or TR_BEGIN_CON.

ASSOC_flags
: Specifies the option flags provided with the primitive. See "Flags" below. Some flags may be provider specific.

OPT_length
: Specifies the length of the protocol options associated with the user data transfer. Supplying protocol options with the primitive is optional. If the transaction user does not provide protocol options with the primitive, the OPT_length and OPT_offset fields must be set to zero (0) by the transaction user. The format of the protocol options are provider specific.

OPT_offset
: Specifies the offset from the beginning of the M_PROTO message block where the protocol options begin. Alignment of the protocol options in the M_PROTO message block is not guaranteed. However, the alignment of the protocol options in the M_PROTO message block are the same as was specified by the transport user.

Flags

TR_MORE_DATA_FLAG
: When set, the MORE_DATA_FLAG indicates that the next TR_CONT_REQ primitive (TIDU) is also part of this TSDU.

TR_RC_FLAG
: By setting this flag on the TR_CONT_REQ, the originating transaction user can request confirmation of receipt of the TR_CONT_REQ primitive.

`TR_SEQ_ASSURANCE`
: By setting this flag on the primitive, the originating transaction user can indicate that "sequence assured" service is requested from the underlying network service provider.

`TR_NO_PERMISSION`
: By setting this flag on the `TR_CONT_REQ`, the originating transaction user can either deny (set) or grant (clear) permission for the transaction peer to terminate the transaction association upon receipt of the corresponding `TR_CONT_IND` primitive. This flag is only used for transaction providers that support this feature (see [Addendum for ANSI Conformance], page 85).

Valid State

This primitive is valid in transaction state `TRS_DATA_XFER`. This primitive is only valid in connection-oriented mode.

New State

The new state for the transaction remains unchanged.

Acknowledgements

This primitive does not require acknowledgement. If a non-fatal error occurs, it is the responsibility of the peer ASE to report it within the upper-layer protocol or using the `TR_ABORT_IND` primitive (see Section 4.2.3.4 [Transaction Abort Indication], page 66). Fatal errors are indicated with the `M_ERROR` message type which results in the failure of all operating system service routines on the *stream*. The allowable fatal errors are as follows:

[`EPROTO`] This error indicates on of the following unrecoverable protocol conditions:

- The transaction interface was found to be in an incorrect state.
- The amount of transaction user data associated with the primitive is outside the range supported by the transaction provider (as specified by the `TIDU_size` parameter of the `TR_INFO_ACK` primitive described in Section 4.1.1.2 [Transaction Information Acknowledgement], page 24.)
- The options requested are either not support by the transaction provider or their use is not specified with the `TR_BEGIN_REQ` primitive.
- The `M_PROTO` message block was not follows by one or more `M_DATA` message blocks.
- The amount of transaction user data associated with the current NSDU is outside the range supported by the transaction provider (as specified by the `TSDU_size` parameter in the `TR_INFO_ACK` primitive described in Section 4.1.1.2 [Transaction Information Acknowledgement], page 24.)
- The `TR_RC_FLAG` and `TR_MORE_DATA_FLAG` were both set in the primitive, or the flags field contained an unknown value.

NOTE: If the interface is in the `TRS_IDLE` state when the provider receives the `TR_CONT_REQ` primitive, then the transaction provider should discard the request without generating a fatal error.

4.2.2.2 Transaction Continue Indication

`TR_CONT_IND`

This transaction provider originated primitive indicates to the transaction user that this message contains transaction user data. As in the `TR_CONT_REQ` primitive (see Section 4.2.2.1 [Transaction Continue Request], page 55), the TSDU can eb segmented into more than one TIDU. The TIDUs are assocated with the TSDU by using the `TR_MORE_DATA_FLAG`. The `TR_RC_FLAG` and `TR_NO_PERMISSION` flags are allowed to be set only on the last TIDU. Use of the `M_PROTO` message blocks is optional (see guidelines describe in see Section 4.2.2.1 [Transaction Continue Request], page 55).

Format

The format of the message is one `M_PROTO` message block, followed by zero or more `M_DATA` message blocks containing user data for the association, structured as follows:

```
typedef struct TR_cont_ind {
    ulong PRIM_type;          /* Always TR_CONT_IND */
    ulong TRANS_id;           /* Transaction id */
    ulong ASSOC_flags;        /* Association flags */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_cont_ind_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type: always `TR_CONT_IND`.

`TRANS_id`
: Indicates the transaction identifier previously indicated by the transport provider to uniquely identify the transaction. The transaction identifier must be indicated by the transaction provider. The transaction identifier must be the same as the transaction identifier that was indicated in the corresponding `TR_BEGIN_IND` or `TR_BEGIN_CON`.

`ASSOC_flags`
: Specifies the option flags provided with the primitive. See “Flags” below. Some flags may be provider specific.

`OPT_length`
: Indicates the length of the protocol options associated with the user data transfer. Protocol options are only indicated by the transaction provider when they were supplied by the underlying protocol. If the transport provider does not indicate protocol options, the `OPT_length` and `OPT_offset` fields must be set to zero (0). The format of the protocol options are provider specific.

`OPT_offset`
: Indicates the offset from the beginning of the `M_PROTO` message block where the protocol options begin.

Flags

`TR_MORE_DATA_FLAG`
: When set, indicates taht the next `TR_CONT_IND` message (TIDU) is part of this TSDU.

`TR_RC_FLAG`
: The value of the flag may indicate either that confirmation is requested or that it is not requested. The flag is allowed to be set only if use of the Receipt Confirmation was agreed between both the transaction users and the transaction provider during transaction association establishment. The value of this flag is always identical to that supplied in the corresponding `TR_CONT_REQ`.

`TR_NO_PERMISSION`
: The value of this flag may indicate either that the transaction peer gives permission (clear) to end the transaction association or does not give permission (set) to end the transaction association. This flag is only valid for transaction providers that support it (see [Addendum for ANSI Conformance], page 85).

Valid State

This primitive is valid in transaction state `TRS_DATA_XFER`. This primitive is only valid in connection-oriented mode.

New State

The new state for the transaction is unchanged.

Rules

— When the `TR_NO_PERMISSION` flag is set, the transaction user must not issue a `TR_END_REQ` primitive in response to this indication.

4.2.3 Transaction Termination

4.2.3.1 Transaction End Request

TR_END_REQ

Format

The format of the message is one M_PROTO message block, followed by zero or more M_DATA message blocks containing user data for the association, structured as follows:

```
typedef struct TR_end_req {
    ulong PRIM_type;          /* Always TR_END_REQ */
    ulong TRANS_id;           /* Transaction id */
    ulong TERM_scenario;      /* Termination scenario */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_end_req_t;
```

Parameters

The primitive has the following arguments:

PRIM_type
: Specifies the primitive type: always TR_END_REQ.

TRANS_id
: Specifies the transaction identifier previously indicated by the transport provider to uniquely identify the transaction. The transaction identifier must be specified by the transaction user unless there is only one transaction supported by the stream in transaction state TRS_DATA_XFER. When specified, the transaction identifier must be the same as the transaction identifier that was indicated by the transaction provider in the corresponding TR_BEGIN_IND or TR_BEGIN_CON.

TERM_scenario
: Specifies the termination scenario. Termination scenarios are provider specific.

OPT_length
: Specifies the length of the protocol options associated with the transaction association termination. Supplying protocol options with the primitive is optional. If the transaction user does not provide protocol options with the primitive, the OPT_length and OPT_offset fields must be set to zero (0) by the transaction user. The format of the protocol options are provider specific.

OPT_offset
: Specifies the offset from the beginning of the M_PROTO message block where the protocol options begin. Alignment of the protocol options in the M_PROTO message block is not guaranteed. However, the alignment of the protocol options in the M_PROTO message block are the same as was specified by the transport user.

Valid State

This primitive is valid in transaction state `TRS_DATA_XFER`. This primitive is only valid in connection-oriented mode.

New State

The new state of the transaction is `TRS_IDLE`.

Rules

Acknowledgements

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

— *Successful:* Correct acknowledgement of the primitive is indicated with the `TR_OK_ACK` primitive described in Section 4.1.4.1 [Transaction Successful Receipt Acknowledgement], page 41.

— *Non-fatal errors:* These errors will be indicated with the `TR_ERROR_ACK` primitive described in Section 4.1.4.2 [Transaction Error Acknowledgement], page 42. The allowable errors are as follows:

`TROUTSTATE`
: The primitive would place the transaction interface out of state for the indicated transaction.

`TRSYSERR`
: A system error occurred and the UNIX System error is indicated in the primitive.

4.2.3.2 Transaction End Indication

TR_END_IND

Format

The format of the message is one M_PROTO message block, followed by zero or more M_DATA message blocks containing user data for the association, structured as follows:

```
typedef struct TR_end_ind {
    ulong PRIM_type;          /* Always TR_END_IND */
    ulong CORR_id;            /* Correlation id */
    ulong TRANS_id;           /* Transaction id */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_end_ind_t;
```

Parameters

The primitive has the following arguments:

PRIM_type
: Indicates the primitive type: always TR_END_IND.

CORR_id
: Indicates the correlation identifier previously specified by the transport user to uniquely identify an outstanding transaction request that has not yet received transaction confirmation. For all other cases, this field must be set to zero (0).

TRANS_id
: Indicates the transaction identifier previously indicated by the transport provider to uniquely identify the transaction. The transaction identifier must be indicated by the transaction provider. The transaction identifier must be the same as the transaction identifier that was indicated in the corresponding TR_BEGIN_IND or TR_BEGIN_CON (if any).

OPT_length
: Indicates the length of the protocol options associated with the transaction association termination. Protocol options are only indicated by the transaction provider when they were supplied by the underlying protocol. If the transport provider does not indicate protocol options, the OPT_length and OPT_offset fields must be set to zero (0). The format of the protocol options are provider specific.

OPT_offset
: Indicates the offset from the beginning of the M_PROTO message block where the protocol options begin.

Valid State

This primitive is valid in transaction states TRS_WCON_CREQ or TRS_DATA_XFER. This primitive is only valid in connection-oriented mode.

New State

The new state for the transaction is TRS_IDLE.

Rules

The following rules apply to the issuance of this primitive:

— This primitive may be issued in response to a TR_BEGIN_REQ primitive. When issued in this case, the transaction provider is indicating that a transaction is both confirmed and terminated.

— This primitive may be issued after receiving a TR_BEGIN_RES or issuing a TR_BEGIN_CON, but before receiving a TR_END_REQ or TR_ABORT_REQ primitive, or issuing a TR_UABORT_IND or TR_PABORT_IND primitive.

— When issued, this primitive indicates the tear-down of the transaction association corresponding to the TRANS_id indicated in the primitive.

4.2.3.3 Transaction User Abort Request

TR_ABORT_REQ

Format

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

```
typedef struct TR_abort_req {
    ulong PRIM_type;          /* Always TR_ABORT_REQ */
    ulong TRANS_id;           /* Transaction id */
    ulong ABORT_cause;        /* Cause of the abort */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_abort_req_t;
```

Parameters

The primitive has the following arguments:

PRIM_type
: Specifies the primitive type: always TR_ABORT_REQ.

TRANS_id
: Specifies the transaction identifier previously indicated by the transport provider to uniquely identify the association. The transaction identifier must be the same as the transaction identifier that was indicated by the transaction provider in the corresponding TR_BEGIN_IND or TR_BEGIN_CON primitive.

ABORT_cause
: Specifies the (user) cause for the abort. Abort causes are provider specific.

OPT_length
: Specifies the length of the protocol options associated with the abort. Supplying protocol options with the primitive is optional. If the transaction user does not provide protocol options with the primitive, the OPT_length and OPT_offset fields must be set to zero (0) by the transaction user. The format of the protocol options are provider specific.

OPT_offset
: Specifies the offset from the beginning of the M_PROTO message block where the protocol options begin. Alignment of the protocol options in the M_PROTO message block is not guaranteed. However, the alignment of the protocol options in the M_PROTO message block are the same as was specified by the transport user.

Modes

This primitive is only valid in connection-oriented mode.

Originator

Transaction user.

Valid State

This primitive is valid in any connection oriented transaction state other than `TRS_IDLE`.

New State

The new state for the transaction is `TRS_IDLE`.

Acknowledgements

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

— *Successful:* Correct acknowledgement of the primitive is indicated with the `TR_OK_ACK` primitive described in Section 4.1.4.1 [Transaction Successful Receipt Acknowledgement], page 41.

— *Non-fatal errors:* These errors will be indicated with the `TR_ERROR_ACK` primitive described in Section 4.1.4.2 [Transaction Error Acknowledgement], page 42. The allowable errors are as follows:

`TRBADDATA`
: The amount of user data specified was invalid.

`TRBADID`
: The transaction identifier specified in the primitive was incorrect or invalid.

`TRNOTSUPPORT`
: This primitive is not supported by the transaction provider.

`TROUTSTATE`
: The primitive would place the transaction interface out of state for the indicated transaction.

`TRSYSERR`
: A system error occurred and the UNIX System error is indicated in the primitive.

The transport provider should not generate an error if it receives this primitive in the `TRS_IDLE` state for the transaction.

4.2.3.4 Transaction Abort Indication

TR_ABORT_IND

This primitive indicates to the user that either a request for association has been denied or an existing association has been aborted.

Format

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

```
typedef struct TR_abort_ind {
    ulong PRIM_type;          /* Always TR_ABORT_IND */
    ulong CORR_id;            /* Correlation id */
    ulong TRANS_id;           /* Transaction id */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
    ulong ABORT_cause;        /* Cause of the abort */
    ulong ORIGINATOR;         /* Originator P or U */
} TR_abort_ind_t;
```

Parameters

The primitive has the following arguments:

PRIM_type
: Indicates the primitive type: always TR_ABORT_IND.

CORR_id
: Indicates the correlation identifier previously specified by the transport user to uniquely identify an outstanding transaction request that has not yet received transaction confirmation. For all other cases, this field must be set to zero (0).

TRANS_id
: Indicates the transaction identifier previously indicated by the transport provider to uniquely identify the transaction. The transaction identifier must be indicated by the transaction provider. The transaction identifier must be the same as the transaction identifier that was indicated in the corresponding TR_BEGIN_IND or TR_BEGIN_CON primitive (if any).

OPT_length
: Indicates the length of the protocol options associated with the transaction association termination. Protocol options are only indicated by the transaction provider when they were supplied by the underlying protocol. If the transport provider does not indicate protocol options, the OPT_length and OPT_offset fields must be set to zero (0). The format of the protocol options are provider specific.

OPT_offset
: Indicates the offset from the beginning of the M_PROTO message block where the protocol options begin.

ABORT_cause
: Indicates the cause of the abort. Abort causes are provider specific.

`ORIGINATOR`
: Indicates the originator of the abort. This field can have values `TR_USER` or `TR_PROVIDER` or `TR_UNSPECIFIED`.

Modes

This primitive is only valid in connection-oriented mode.

Originator

Transaction provider.

Valid State

This primitive is valid in any connection oriented transaction state other than `TRS_IDLE`.

New State

The new state for the transaction is `TRS_IDLE`.

4.3 Connectionless Mode Primitives

4.3.1 Transaction Phase

4.3.1.1 Transaction Unidirectional Request

`TR_UNI_REQ`

This primitive requests that the TR provider send the specified unidirectional (connectionless) message to the specified destination with the specified options and optional originating protocol address.

Format

The format of the message is one `M_PROTO` message block, followed by zero or more `M_DATA` message blocks containing user data for the association, structured as follows:

```
typedef struct TR_uni_req {
    ulong PRIM_type;          /* Always TR_UNI_REQ */
    ulong DEST_length;        /* Destination address length */
    ulong DEST_offset;        /* Destination address offset */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
    ulong ORIG_length;        /* Originating address length */
    ulong ORIG_offset;        /* Originating address offset */
} TR_uni_req_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Specifies the primitive type: always `TR_UNI_REQ`.

`DEST_length`
: Specifies the length of the protocol address to which to send the unidirectional invocation.

`DEST_offset`
: Specifies the offset from the beginning of the `M_PROTO` message block where the protocol address begins.

`ORIG_length`
: Specifies the length of the protocol address from which to send the unidirectional invocation. Specification of the originating protocol address (`ORIG_length` and `ORIG_offset`) is optional. When not specified the TR provider will implicitly associate the local protocol address used in the bind service (see Section 4.1.2.1 [Transaction Bind Request], page 26) with the primitive as the originating protocol address.

`ORIG_offset`
: Specifies the offset from the beginning of the `M_PROTO` message block where the protocol address begins.

`OPT_length`
: Specifies the length of the protocol options associated with the unidirectional invocation.

`OPT_offset`
: Specifies the offset from the beginning of the `M_PROTO` message block where the protocol options begin.

Valid State

This primitive is valid in state `TRS_IDLE`. This primitive is only valid in connectionless mode.

New State

The new state remains unchanged.

Rules

Acknowledgements

This primitive does not require an acknowledgement.[1] If a non-fatal error occurs, it is the responsibility of the TR provider to report it with the `TR_NOTICE_IND` indication. Fatal errors are indicated with the `M_ERROR` message type which results in the failure of all operating system service routines on the *stream*. The allowable fatal errors are as follows:

[`EPROTO`] This error indicates one of the following unrecoverable protocol conditions:

- The TR service interface was found to be in an incorrect state.
- The amount of TR user data associated with the primitive defines an APDU (ACSE Protocol Data Unit) larger than that allowed by the TR provider.

[1] This is a TCAP operations class 4 or a ROSE operations class 5 transaction that requires neither a positive or negative acknowledgement.

4.3.1.2 Transaction Unidirectional Indication

`TR_UNI_IND`

This primitive indicates to the TR user that a unidirectional invocation has been received from the specified source address.

Format

The format of the message is one `M_PROTO` message block, followed by zero or more `M_DATA` message blocks containing user data for the association, where each `M_DATA` message block contains at least one byte of data, structured as follows:

```
typedef struct TR_uni_ind {
    ulong PRIM_type;          /* Always TR_UNI_REQ */
    ulong DEST_length;        /* Destination address length */
    ulong DEST_offset;        /* Destination address offset */
    ulong ORIG_length;        /* Originating address length */
    ulong ORIG_offset;        /* Originating address offset */
    ulong OPT_length;         /* Options structure length */
    ulong OPT_offset;         /* Options structure offset */
} TR_uni_ind_t;
```

Parameters

The primitive has the following arguments:

`PRIM_type`
: Indicates the primitive type: always `TR_UNI_IND`.

`DEST_length`
: Indicates the length of the protocol address to which the message was sent. This is not necessarily the same as the local protocol address to which the *stream* is bound. The address provided here may contain additional information for some protocols. So, for example, under TCAP, although the *stream* is bound to an SCCP subsystem, this protocol address may contain the SCCP Global Title.

`DEST_offset`
: Indicates the offset from the start of the `M_PROTO` message block where the protocol address begins.

`ORIG_length`
: Indicates the length of the protocol address from which the message was sent.

`ORIG_offset`
: Indicates the offset from the start of the `M_PROTO` message block where the protocol address begins.

`OPT_length`
: Indicates the length of the protocol options that were associated with the received message.

`OPT_offset`
: Indicates the offset from the start of the `M_PROTO` message block where the protocol options begin.

Valid State

This primitive is only issued in state `TRS_IDLE`. This primitive is only valid in connectionless mode.

New State

The new state remains unchanged.

Rules

The proper alignment of the destination address, originating address and protocol options in the `M_PROTO` message block is not guaranteed.

4.3.1.3 Transaction Notice Indication

TR_NOTICE_IND

This primtiive indicates to the transaction user that a component of a transaction produced an error.

Format

The format of the message is one M_PCPROTO message block, followed by zero or more M_DATA message blocks containing user data for the association, structured as follows:

```
typedef struct TR_notice_ind {
    ulong PRIM_type;          /* Always TR_NOTICE_IND */
    ulong CORR_id;            /* Correlation id */
    ulong TRANS_id;           /* Transaction id */
    ulong REPORT_cause;       /* SCCP return cause */
} TR_notice_ind_t;
```

Parameters

The primitive has the following arguments:

PRIM_type
: Indicates the primitive type. Always TR_NOTICE_IND.

CORR_id
: Indicates the transaction user assigned transaction identifier.

TRANS_id
: Indicates the transaction provider assigned transaction identifier.

REPORT_cause
: Indicates the defined protocol dependent error code.

Modes

This primitive is only issued in Operations Classes that provide negative acknowledgements.

Originator

This primitive is originated by the TR provider.

Valid State

This primitive is only valid in connectionless mode.

New State

The new state remains unchanged.

Rules

5 Diagnostics Requirements

There are two error handling facilities available to the TR user: one to handle non-fatal errors and one to handle fatal errors.

5.1 Non-Fatal Errors

The non-fatal errors are those that a TR user can correct, and are reported in the form of an error acknowledgement to the appropriate primitive in error. Only those primitive which require acknowledgements may generate a non-fatal error acknowledgement. These acknowledgements always report syntactical error in the specified primitive when the TR provider receives the primitive. The primitive descriptions[1] define those primitive and rules regarding acknowledgement for each primitive. These errors are reported to the TR user with the `TR_ERROR_ACK` primitive, (see Section 4.1.4.2 [Transaction Error Acknowledgement], page 42), and give the TR user the option of reissuing the TR service primitive that cause the error. The `TR_ERROR_ACK` primitive also indicates to the TR user that no action was taken by the TR provider upon receipt of the primitive which cause the error.

These errors do not change the state of the TR service interface as seen by the TR user. The state of the interface after the issuance of a `TR_ERROR_ACK` primitive should be the same as it was before the TR provider receive the interface primitive that was in error.

The allowable errors that can be reported on the receipt of a TR initiated primitive are presented in the description of the appropriate primitives, see Chapter 4 [TRI Primitives], page 21.

5.2 Fatal Errors

Fatal errors are those that cannot be corrected by the TR user, or those errors that result in an uncorrectable error in the interface or in the TR provider.

The most common of these errors are listed under the appropriate primitives (see Chapter 4 [TRI Primitives], page 21). The transaction provider should issue fatal errors only if the transaction user cannot correct the condition that caused the error or if the transaction provider has no means of reporting a transaction user correctable error. If the transaction provider detects an uncorrectable non-protocol error internal to the transaction provider, the provider should issue a fatal error to the user.

Fatal errors are indicated to the transaction user with the *STREAMS* message type `M_ERROR` with the UNIX System error [`EPROTO`]. This is the only type of error that the transaction provider should use to indicate a fatal protocol error to the transaction user. The message `M_ERROR` will result in the failure of all the operating system service routines on the *stream*. The only way for the user to recover from a fatal error is to ensure that all processes close the file associated with the *stream*. Then the user may reopen the file associated with the *stream*.

[1] See Chapter 4 [TRI Primitives], page 21.

6 Transaction Service Interface Sequence of Primitives

The allowable sequence of primitives are described in the state diagrams and tables for both the connection-oriented and connectionless mode mode transaction services described in Appendix B [State/Event Tables], page 99.

6.1 Rules for State Maintenance

6.1.1 General Rules for State Maintenace

The following are rules regarding the maintenance of the state of the interface:

- It is the responsibility of the transaction provider to keep record of the state of the interface as viewed by the transaction user.
- The transaction provider must never issue a primitive that places the interface out of state.
- The uninitialized state of a *stream* is the initial and final state, and it must be bound (see Section 4.1.2.1 [Transaction Bind Request], page 26) before the transaction provider may view it as an active *stream*.
- If the transaction provider sends a `M_ERROR` upstream, it should also drop any further messages received on its write side of the *stream*.

6.1.2 Connection-Oriented Transaction Service Rules for State Maintenace

The following rules apply only to the connection-oriented mode transaction services:

- A transaction association end procedure can be initiated at any time during the transaction association establishment or user data transfer phases.
- The state tables for the connection-oriented mode transaction service providers include the management of the correlation and transaction identifiers when a transaction provider sends multiple `TR_BEGIN_IND` indications or accepts multiple `TR_BEGIN_REQ` requests without waiting for the response or confirmation to the previous indication or request. It is the responsibility of the transaction provider not to change state until all the indications or requests have been responded to or confirmed, therefore the provider should remain in the `TRS_WRES_CIND` or `TRS_WACK_CREQ` state while there are any outstanding begin indications or requests pending response or confirmation. The provider should change state appropriately when all the begin indications or requests have been responded to or confirmed.
- The only time the state of the transaction service interface of a *stream* may be transferred to another *stream* is when it is indicated in a `TR_BEGIN_RES` primitive. The following rules then apply to the cooperating *streams*:
  - The *stream* that is to accept the current state of the interface must be bound to an appropriate protocol address and must be in the idle state.[1]

[1] This is not really true for either TRI or TPI. The accepting stream can be bound or unbound, and for some protocols may be bound to an address different or the same as the stream upon which the begin indication was issued.

— The user transferring the current state of a *stream* must have the correct permissions for the use of the protocol address bound to the accepting *stream*.

— The *stream* which transfers the state of the transaction interface must be placed into an appropriate state after the completion of the transfer.

6.2 Rules for Precedence of Primitives on a *Stream*

6.2.1 General Rules for Precedence of Primitives

The following rules apply to the precedence of transaction interface primitives with respect to their position on a *stream*:[2]

- The transaction provider has responsibility for determining precedence of its *stream* write queue, as per the rules defined in Appendix C [Primitive Precedence Tables], page 101. The appendix specifies the rules for precedence for both the connection-oriented and connectionless transaction services.
- The transaction user has the responsibility for determining precedence on its *stream* read queue, as per the rules defined in Appendix C [Primitive Precedence Tables], page 101.
- All primitives on the *stream* are assumed to be placed on the queue in the correct sequence as defined above.

6.2.2 Connection-Oriented Transaction Service Rules for Precedence of Primitives

The following rules apply only to the connection-oriented transaction services:

- There is no guarantee of delivery of user data once a `TR_ABORT_REQ` primitive has been issued.

6.3 Rules for Flushing Queues

6.3.1 General Rules for Flushing Queues

The following rules pertain to flushing of *stream* queues: (No other flushes should be needed to keep the queues in the proper condition.)

- The transaction providers must be aware that they will receive `M_FLUSH` message from upstream. These flush requests are issued to ensure that the providers receive certain messages and primitives. It is the responsibility of the providers to act appropriately as deemed necessary by the providers.
- The transaction provider must send up a `M_FLUSH` message to flush both the read and write queues after receiving a successful `TR_UNBIND_REQ` message and prior to issuing the `TR_OK_ACK` primitive.

[2] The *stream* queue which contains a transaction user initiated primitives is referred to as the *stream* write queue. The *stream* queue which contains the transaction provider initiated primitives is referred to as the *stream* read queue.

6.3.2 Connection-Oriented Transaction Service Rules for Flushing Queues

The following rules apply only to the connection-oriented transaction services:

- If the interface is in the TRS_DATA_XFER, TRS_WIND_ORDREL or TRS_WACK_ORDREL state, the transaction provider must send up a M_FLUSH message to flush both the read and write queues before sending up a TR_ABORT_IND.
- If the interface is in the TRS_DATA_XFER, TRS_WIND_ORDREL or TRS_WACK_ORDREL state, the transaction provider must send up a M_FLUSH message to flush both the read and write queues after receiving a successful TR_ABORT_REQ primitive and before issuing the TR_OK_ACK primitive.

Addendum for ITU-T Conformance

This section describes the formats and rules that are specified to ITU-T Q.771 operation. The addendum must be used along with the generic TRI as defined in the main document when implementing a TR provider that will be configured with the ITU-T Q.771 (TCAP) Transaction Sub-Layer.

Quality of Service: Model and Description

The "Quality of Service" characteristics apply to both connection-oriented and connectionless transaction services.

QoS Overview

QoS (Quality of Service) is described in terms of QoS parameters. There are two types of QoS parameters:

1. Those that are "negotiated" on a per-association basis during transaction association establishment.[1]
2. Those that are not "negotiated" and their values are selected or determined by local management methods.

TRI Primitives: Rules for ITU-T Q.771 Conformance

The following rules apply to the TRI primitives for ITU-T Q.771 (TCAP) compatibility:

Addressing

TCAP uses SCCP formatted addresses instead of ISO Presentation Layer addresses.

Address Format

The address format for a TCAP address is as follows:

Options

TCAP Level Options

Application Context Name

User Information

SCCP Level Options

SCCP Quality of Service Options

The TCAP interface uses protocol level `T_SS7_SCCP` for options at the SCCP level. SCCP QoS parameters are communicated to the underlying transaction provider using the option name `T_SCCP_QOS`. There are three QoS structure that can be used in this fashion as follows:

[1] The connectionless transaction services do not support end-to-end QoS parameter negotiation.

Option Name	Option Type	Meaning
T_SCCP_QOS	N_qos_sel_sccp_t	For use with TR_UNI_REQ, TR_BEGIN_REQ, TR_BEGIN_RES, TR_CONT_REQ, TR_END_REQ, TR_ABORT_REQ.
T_SCCP_QOS	N_qos_opt_sel_sccp_t	For use with TR_BEGIN_REQ, TR_BEGIN_RES.
T_SCCP_QOS	N_qos_range_sccp_t	For use with TR_INFO_ACK.

Quality of service struct *N_qos_sel_sccp_t* has the following fields:

n_qos_type
: This is the NPI Quality of Service structure type and is always set to `N_QOS_SEL_SCCP`, `N_QOS_OPT_SEL_SCCP`, or `N_QOS_RANGE_SCCP`.

protocol_class
: This is the protocol class. The *protocol_class* field can be one of the following:
  - `N_QOS_PCLASS_0` (SCCP connectionless protocol class 0),
  - `N_QOS_PCLASS_1` (for SCCP connectionless protocol class 1),
  - `N_QOS_PCLASS_2` (for SCCP connection-oriented protocol class 2),
  - `N_QOS_PCLASS_3` (for SCCP connection-oriented protocol class 3) or
  - `QOS_UNKNOWN`.

  `N_QOS_PCLASS_2` and `N_QOS_PCLASS_3` are not applicable to TCAP.

option_flags
: If the *options_flags* field has bit `N_QOS_OPT_RETERR` set then the SCCP will return the PDU on error.

importance
: This is the importance of the message for consideration for SCCP flow control. This value is not normally set by the user. It can be any integer number from 0 to 7, or `QOS_UNKNOWN`.

sequence_selection
: This affects the SLS (Signalling Link Selection) value that will be used for protocol classes `N_QOS_PCLASS_0` and `N_QOS_PCLASS_1`. This value is not normally set by the user and can be an integer value or `QOS_UNKNOWN`.

message_priority
: This affects the MP (Message Priority) value that will be used for specific messages in all protocol classes. This value is not normally set by the use and can be any integer value from 0 to 3 or the value `QOS_UNKNOWN`.

Supported Services

Common Transaction Services

Information Service

`TR_INFO_REQ`

`TR_INFO_ACK`

Parameters

The following discusses the values which may be returned in a `TR_INFO_ACK` primitive in response to a `TR_INFO_REQ` primitive.

`ASDU_size`

Depending on the underlying SCCP layer, TCAP can have effectively no limit to the amount of user data that can be sent in a particular transaction. Protocol variants or versions of SCCP that support XUDT and segmentation-reassembly of protocol class 0 or 1 messages will set `ASDU_size` to `T_INFINITE` (‘`-1`’). For protocol variants of SCCP or other underlying network providers that do not support segmentation/reassembly of long messages, the provider wills et `ASDU_size` to the maximum size (number of octets) of user data that can be guaranteed transferred when associated with a single `TR_BEGIN_RES` or `TR_CONT_REQ` message.

`EASDU_size`

TCAP has no expedited data service and the value of `EASDU_size` is set to `T_UNKNOWN` (‘`-2`’).

`CDATA_size`

TCAP can send user data with the initial Begin (Query) or first Continue (Conversation) package and can also send *Application Context* and *User Information* in either package. These messages correspond to *TR-BEGIN* and the first *TR-CONTINUE* after receiving a *TR-BEGIN* and they correspond to `TR_BEGIN_REQ` and `TR_BEGIN_RES`. Because the underlying SCCP connectionless network may support unlimited size NSDUs, this value may be set to `T_INFINITE` (‘`-1`’) or may be set to the maximum amount of user data (including *Application Context*, *User Information* and user data) that can be sent or received in either package. This informs the user as to what size to make data buffers associated with transaction begin indications and confirmations (`TR_BEGIN_IND`, `TR_BEGIN_CON`) and how much data can be sent with transaction begin requests and responses (`TR_BEGIN_REQ`, `TR_BEGIN_RES`).

`DDATA_size`

TCAP can send transaction end data (user data) with the final End (Response) package. These messages correspond to the *TR-END* primitive and the `TR_END_REQ` or `TR_END_IND`. Again, because the underlying SCCP connectionless network may support unlimited size NSDUs, this value may be set to `T_INFINITE` (‘`-1`’) or may be set to the maximum amount of transaction end data that can be sent or received in the End (Response) package. This informs the user as to what size to make data buffers associated with transaction end indications (`TR_END_IND`) and how much data can be sent with transaction end requests (`TR_END_REQ`).

`ADDR_size`
: This is the maximum TCAP address size that can be communicated across the interface. This address size is the maximum size of the defined SCCP address structure (‘`sizeof sccp_addr_t`’) that also will include address digits up to a maximum of `SCCP_MAX_ADDR_LENGTH` octets of digits. This informs the user as to what size it should reserver for control buffers so as to receive control information without buffer truncation.

`OPT_size`
: This is the maximum size of the options field used in any TRI message (see Chapter 4 [TRI Primitives], page 21) and is the sum of the maximum option sizes of one of each of the options that can occur together. This informs the user as to what size it should reserve for control buffers to ensure that received control messages that include options cna be contained within the buffer without truncation.

`TIDU_size`
: Although a TCAP provider can support unlimited ASDU size, it cannot normally support unlimited TIDU size. This is because the underlying SCCP NSDU may be limited in size. The TCAP provider is not responsible for segmenting user data sequences offered to the provider from the user in an `M_DATA` message chain. This is the maximum size of the TIDU which corresponds to the maximum size of the underlying NSDU. Because the underlying SCCP provider may have no limit on the NSDU size (i.e, it supports segmentation of connectionless NSDUs) this may be more in the manner of a optimal recommendation to the user rather than an absolute maximum. Because of this, a given TCAP provider might not reject TIDUs which are larger than this value.

`SERV_type`
: There are two service types supported by a transaction provider: connection-oriented transaction service (COTS) and connectionless transaction service (CLTS). CLTS is a connectionless unidirectional transaction service with no error notification. COTS is a connection-oriented transaction services with or without error notification. The value reflected here is dependent on the setting of option `T_ACSE_PCLASS` or `T_TCAP_OCLASS`.

`CURRENT_state`
: Provides the current state of the transaction interface. TCAP providers use the same states as other TRI providers.

`PROVIDER_flag`
: Unused.

`TRI_version`
: Set to the current version.

Address service

`TR_ADDR_REQ`

TR_ADDR_ACK

Bind Service

TR_BIND_REQ

TR_BIND_ACK

Options Management Service

TR_OPTMGMT_REQ

TR_OPTMGMT_ACK

Connection-Oriented Transaction Services

Transaction Begin

TR_BEGIN_REQ

TR_BEGIN_IND

TR_BEGIN_RES

TR_BEGIN_CON

Transaction Continue

TR_CONT_REQ

TR_CONT_IND

Transaction End

TR_ABORT_REQ

TR_ABORT_IND

TR_END_REQ

TR_END_IND

Connectionless Transaction Services

TR_UNI_REQ

TR_UNI_IND

TR_NOTICE_IND

Addendum for ANSI Conformance

This section describes the formats and rules that are specified to ANSI T1.114 operation. The addendum must be used along with the generic TRI as defined in the main document when implementing a TR provider that will be configured with the ANSI T1.114 (TCAP) Transaction Sub-Layer.[1]

Quality of Service: Model and Description

The "Quality of Service" characteristics apply to both connection-oriented and connectionless transaction services.

QoS Overview

QoS (Quality of Service) is described in terms of QoS parameters. There are two types of QoS parameters:

1. Those that are "negotiated" on a per-association basis during transaction association establishment.[2]
2. Those that are not "negotiated" and their values are selected or determined by local management methods.

TRI Primitives: Rules for ANSI T1.114 Conformance

The following rules apply to the TRI primitives for ANSI T1.114 (TCAP) compatibility:

Addressing

TCAP uses SCCP formatted addresses instead of ISO Presentation Layer addresses.

Address Format

The address format for a TCAP address is as follows:

Options

TCAP Level Options

Application Context Name

User Information

SCCP Level Options

[1] It should be noted that ANSI T1.114 does not provide a distinction between the TC and TR Sub-Layers of TCAP, and do not specify a TC-User or TR-User interface at all. However, as it is still based on ITU-T Recommendation X.219, there can exist an identifiable TR Sub-Layer interface within ANSI TCAP.

[2] The connectionless transaction services do not support end-to-end QoS parameter negotiation.

SCCP Quality of Service Options

The TCAP interface uses protocol level `T_SS7_SCCP` for options at the SCCP level. SCCP QoS parameters are communicated to the underlying transaction provider using the option name `T_SCCP_QOS`. There are three QoS structure that can be used in this fashion as follows:

Option Name	Option Type	Meaning
T_SCCP_QOS	N_qos_sel_sccp_t	For use with TR_UNI_REQ, TR_BEGIN_REQ, TR_BEGIN_RES, TR_CONT_REQ, TR_END_REQ, TR_ABORT_REQ.
T_SCCP_QOS	N_qos_opt_sel_sccp_t	For use with TR_BEGIN_REQ, TR_BEGIN_RES.
T_SCCP_QOS	N_qos_range_sccp_t	For use with TR_INFO_ACK.

Quality of service struct *N_qos_sel_sccp_t* has the following fields:

n_qos_type
: This is the NPI Quality of Service structure type and is always set to `N_QOS_SEL_SCCP`, `N_QOS_OPT_SEL_SCCP`, or `N_QOS_RANGE_SCCP`.

protocol_class
: This is the protocol class. The *protocol_class* field can be one of the following:

  - `N_QOS_PCLASS_0` (SCCP connectionless protocol class 0),
  - `N_QOS_PCLASS_1` (for SCCP connectionless protocol class 1),
  - `N_QOS_PCLASS_2` (for SCCP connection-oriented protocol class 2),
  - `N_QOS_PCLASS_3` (for SCCP connection-oriented protocol class 3) or
  - `QOS_UNKNOWN`.

  `N_QOS_PCLASS_2` and `N_QOS_PCLASS_3` are not applicable to TCAP.

option_flags
: If the *options_flags* field has bit `N_QOS_OPT_RETERR` set then the SCCP will return the PDU on error.

importance
: This is the importance of the message for consideration for SCCP flow control. This value is not normally set by the user. It can be any integer number from 0 to 7, or `QOS_UNKNOWN`.

sequence_selection
: This affects the SLS (Signalling Link Selection) value that will be used for protocol classes `N_QOS_PCLASS_0` and `N_QOS_PCLASS_1`. This value is not normally set by the user and can be an integer value or `QOS_UNKNOWN`.

message_priority
: This affects the MP (Message Priority) value that will be used for specific messages in all protocol classes. This value is not normally set by the use and can be any integer value from 0 to 3 or the value `QOS_UNKNOWN`.

Supported Services

Common Transaction Services

Information Service

TR_INFO_REQ

TR_INFO_ACK

Parameters

The following discusses the values which may be returned in a TR_INFO_ACK primitive in response to a TR_INFO_REQ primitive.

ASDU_size

Depending on the underlying SCCP layer, TCAP can have effectively no limit to the amount of user data that can be sent in a particular transaction. Protocol variants or versions of SCCP that support XUDT and segmentation-reassembly of protocol class 0 or 1 messages will set ASDU_size to T_INFINITE ('-1'). For protocol variants of SCCP or other underlying network providers that do not support segmentation/reassembly of long messages, the provider wills et ASDU_size to the maximum size (number of octets) of user data that can be guaranteed transferred when associated with a single TR_BEGIN_RES or TR_CONT_REQ message.

EASDU_size

TCAP has no expedited data service and the value of EASDU_size is set to T_UNKNOWN ('-2').

CDATA_size

TCAP can send user data with the initial Begin (Query) or first Continue (Conversation) package and can also send *Application Context* and *User Information* in either package. These messages correspond to *TR-BEGIN* and the first *TR-CONTINUE* after receiving a *TR-BEGIN* and they correspond to TR_BEGIN_REQ and TR_BEGIN_RES. Because the underlying SCCP connectionless network may support unlimited size NSDUs, this value may be set to T_INFINITE ('-1') or may be set to the maximum amount of user data (including *Application Context*, *User Information* and user data) that can be sent or received in either package. This informs the user as to what size to make data buffers associated with transaction begin indications and confirmations (TR_BEGIN_IND, TR_BEGIN_CON) and how much data can be sent with transaction begin requests and responses (TR_BEGIN_REQ, TR_BEGIN_RES).

DDATA_size

TCAP can send transaction end data (user data) with the final End (Response) package. These messages correspond to the *TR-END* primitive and the TR_END_REQ or TR_END_IND. Again, because the underlying SCCP connectionless network may support unlimited size NSDUs, this value may be set to T_INFINITE

('-1') or may be set to the maximum amount of transaction end data that can be sent or received in the End (Response) package. This informs the user as to what size to make data buffers associated with transaction end indications (TR_END_IND) and how much data can be sent with transaction end requests (TR_END_REQ).

ADDR_size
: This is the maximum TCAP address size that can be communicated across the interface. This address size is the maximum size of the defined SCCP address structure ('sizeof sccp_addr_t') that also will include address digits up to a maximum of SCCP_MAX_ADDR_LENGTH octets of digits. This informs the user as to what size it should reserver for control buffers so as to receive control information without buffer truncation.

OPT_size
: This is the maximum size of the options field used in any TRI message (see Chapter 4 [TRI Primitives], page 21) and is the sum of the maximum option sizes of one of each of the options that can occur together. This informs the user as to what size it should reserve for control buffers to ensure that received control messages that include options cna be contained within the buffer without truncation.

TIDU_size
: Although a TCAP provider can support unlimited ASDU size, it cannot normally support unlimited TIDU size. This is because the underlying SCCP NSDU may be limited in size. The TCAP provider is not responsible for segmenting user data sequences offered to the provider from the user in an M_DATA message chain. This is the maximum size of the TIDU which corresponds to the maximum size of the underlying NSDU. Because the underlying SCCP provider may have no limit on the NSDU size (i.e, it supports segmentation of connectionless NSDUs) this may be more in the manner of a optimal recommendation to the user rather than an absolute maximum. Because of this, a given TCAP provider might not reject TIDUs which are larger than this value.

SERV_type
: There are two service types supported by a transaction provider: connection-oriented transaction service (COTS) and connectionless transaction service (CLTS). CLTS is a connectionless unidirectional transaction service with no error notification. COTS is a connection-oriented transaction services with or without error notification. The value reflected here is dependent on the setting of option T_ACSE_PCLASS or T_TCAP_OCLASS.

CURRENT_state
: Provides the current state of the transaction interface. TCAP providers use the same states as other TRI providers.

PROVIDER_flag
: Unused.

TRI_version
: Set to the current version.

Address service

TR_ADDR_REQ

TR_ADDR_ACK

Bind Service

TR_BIND_REQ

TR_BIND_ACK

Options Management Service

TR_OPTMGMT_REQ

TR_OPTMGMT_ACK

Connection-Oriented Transaction Services

Transaction Begin

TR_BEGIN_REQ

TR_BEGIN_IND

TR_BEGIN_RES

TR_BEGIN_CON

Transaction Continue

TR_CONT_REQ

TR_CONT_IND

Transaction End

TR_ABORT_REQ

TR_ABORT_IND

TR_END_REQ

TR_END_IND

Connectionless Transaction Services

TR_UNI_REQ

TR_UNI_IND

TR_NOTICE_IND

Addendum for ETSI Conformance

ETSI Quality of Service Model and Description

QoS Overview

TRI Primitives: Rules for ETSI ETS 300 287 Conformance

Addressing

Address Format

Options

TCAP Level Options

SCCP Level Options

ETSI Supported Services

Common Transaction Services

Information service

`TR_INFO_REQ`

`TR_INFO_ACK`

Address service

`TR_ADDR_REQ`

`TR_ADDR_ACK`

Bind Service

`TR_BIND_REQ`

`TR_BIND_ACK`

Options Management Service

`TR_OPTMGMT_REQ`

`TR_OPTMGMT_ACK`

Connection-Oriented Transaction Services

Transaction Begin

TR_BEGIN_REQ

TR_BEGIN_IND

TR_BEGIN_RES

TR_BEGIN_CON

Transaction Continue

TR_CONT_REQ

TR_CONT_IND

Transaction End

TR_ABORT_REQ

TR_ABORT_IND

TR_END_REQ

TR_END_IND

Connectionless Transaction Services

TR_UNI_REQ

TR_UNI_IND

TR_NOTICE_IND

Appendix A Mapping TRI Primitives

A.1 Mapping TRI Primitives to ITU-T Q.771

A.2 Mapping TRI Primitives to ANSI T1.114

A.3 Mapping TRI Primitives to ITU-T X.219

A.3.1 State Mapping

A.3.2 Primitive Mapping

A.3.2.1 A-ASSOCIATE

Request

Indication

Response

Confirm

A.3.2.2 A-RELEASE

Request

Indication

Response

Confirm

A.3.2.3 A-ABORT

Request

Indication

A.3.2.4 A-P-ABORT

Indication

A.3.2.5 A-UNIT-DATA

Request

Indication

A.3.3 Parameter Mapping

Application Context Name

Calling AP Title

Calling AE Qualifier

Calling AP Invocation-identifier

Calling AE Invocation-identifier

Called AP Title

Called AE Qualifier

Called AP Invocation-identifier

Called AE Invocation-identifier

Responding AP Title

Responding AE Qualifier

Responding AP Invocation-identifier

Responding AE Invocation-identifier

User Information

Result

Result Source

Diagnostic

Calling Presentation Address

Called Presentation Address

Responding Presentation Address

Presentation Context Definition List

Presentation Context Definition Result List

Default Presentation Context Name

Default Presentation Context Result

Quality of Service

Session Requirements

Initial Sycnhronization Point Serial Number

Initial Assignment of Tokens

Session-connection Identifier

Reason

User Information

Result

Abort Source

User Information

Provider Reason

Authentication

Authentication-mechanism name

Authentication-value

ACSE Requriements

Diagnostic

Application Context Identifier

Application Context Name List

Appendix B State/Event Tables

Appendix C Primitive Precedence Tables

Appendix D TRI Header File Listing

```
#define TR_INFO_REQ              0      /* Information request */
#define TR_BIND_REQ              1      /* Bind to network address */
#define TR_UNBIND_REQ            2      /* Unbind from network address */
#define TR_OPTMGMT_REQ           5      /* Options management request */
#define TR_UNI_REQ               6      /* Unidirectional request */
#define TR_BEGIN_REQ             7      /* Begin transaction request */
#define TR_BEGIN_RES             8      /* Begin transaction response-Continue request */
#define TR_CONT_REQ              9      /* Continue transaction request */
#define TR_END_REQ              10      /* End transaction request */
#define TR_ABORT_REQ            11      /* Abort transaction request */
#define TR_ADDR_REQ             25      /* Address request */

#define TR_INFO_ACK             12      /* Information acknowledgement */
#define TR_BIND_ACK             13      /* Bound to network address */
#define TR_OK_ACK               15      /* Success acknowledgement */
#define TR_ERROR_ACK            16      /* Error acknowledgement */
#define TR_OPTMGMT_ACK          17      /* Options management acknowledgement */
#define TR_UNI_IND              18      /* Unidirectional indication */
#define TR_BEGIN_IND            19      /* Begin transaction indication */
#define TR_BEGIN_CON            20      /* Begin transaction confirmation-Continue ind */
#define TR_CONT_IND             21      /* Continue transaction indication */
#define TR_END_IND              22      /* End transaction indication */
#define TR_ABORT_IND            23      /* Abort transaction indication */
#define TR_NOTICE_IND           24      /* Error indication */
#define TR_ADDR_ACK             27      /* Address acknowledgement */

#define TR_QOS_SEL1             0x0501

typedef struct {
    t_uscalar_t type;           /* Always TR_QOS_SEL1 */
    t_uscalar_t flags;          /* Return option */
    t_uscalar_t seq_ctrl;       /* Sequence Control */
    t_uscalar_t priority;       /* Message priority */
} TR_qos_sel1_t;

/*
 * TRPI interface states
 */
#define TRS_UNBND       0       /* TR user not bound to network address */
#define TRS_WACK_BREQ   1       /* Awaiting acknowledgement of N_BIND_REQ */
#define TRS_WACK_UREQ   2       /* Pending acknowledgement for N_UNBIND_REQ */
#define TRS_IDLE        3       /* Idle, no connection */
#define TRS_WACK_OPTREQ 4       /* Pending acknowledgement of N_OPTMGMT_REQ */
#define TRS_WACK_RRES   5       /* Pending acknowledgement of N_RESET_RES */
#define TRS_WCON_CREQ   6       /* Pending confirmation of N_CONN_REQ */
#define TRS_WRES_CIND   7       /* Pending response of N_CONN_REQ */
#define TRS_WACK_CRES   8       /* Pending acknowledgement of N_CONN_RES */
#define TRS_DATA_XFER   9       /* Connection-mode data transfer */
#define TRS_WCON_RREQ   10      /* Pending confirmation of N_RESET_REQ */
#define TRS_WRES_RIND   11      /* Pending response of N_RESET_IND */
#define TRS_WACK_DREQ6  12      /* Waiting ack of N_DISCON_REQ */
#define TRS_WACK_DREQ7  13      /* Waiting ack of N_DISCON_REQ */
#define TRS_WACK_DREQ9  14      /* Waiting ack of N_DISCON_REQ */
#define TRS_WACK_DREQ10 15      /* Waiting ack of N_DISCON_REQ */
```

```
#define TRS_WACK_DREQ11 16      /* Waiting ack of N_DISCON_REQ */

#define TRS_NOSTATES    17

/*
 * TR_ERROR_ACK error return code values
 */
#define TRBADADDR       1       /* Incorrect address format/illegal address information */
#define TRBADOPT        2       /* Options in incorrect format or contain illegal
                                   information */
#define TRACCESS        3       /* User did not have proper permissions */
#define TRNOADDR        5       /* TR Provider could not allocate address */
#define TROUTSTATE      6       /* Primitive was issues in wrong sequence */
#define TRBADSEQ        7       /* Sequence number in primitive was incorrect/illegal */
#define TRSYSERR        8       /* UNIX system error occurred */
#define TRBADDATA       10      /* User data spec. outside range supported by TR provider
                                 */
#define TRBADFLAG       16      /* Flags specified in primitive were illegal/incorrect */
#define TRNOTSUPPORT    18      /* Primitive type not supported by the TR provider */
#define TRBOUND         19      /* Illegal second attempt to bind listener or default
                                   listener */
#define TRBADQOSPARAM   20      /* QOS values specified are outside the range supported
                                   by the TR provider */
#define TRBADQOSTYPE    21      /* QOS structure type specified is not supported by the
                                   TR provider */
#define TRBADTOKEN      22      /* Token used is not associated with an open stream */
#define TRNOPROTOID     23      /* Protocol id could not be allocated */

/*
 *  TR_ABORT_IND originator
 */
#define TR_PROVIDER             0x0001
#define TR_USER                 0x0002

/*
 *  TR_ABORT abort causes
 */
#define TR_ABTC_APPL_UNREC_MSG_TYPE             0x0100  /* unrecognized message type */
#define TR_ABTC_APPL_UNREC_TRANS_ID             0x0101  /* unrecognized transaction id */
#define TR_ABTC_APPL_BAD_XACT_PORTION           0x0102  /* badly formatted transaction
                                                           portion */
#define TR_ABTC_APPL_INCORRECT_XACT_PORTION     0x0103  /* incorrect transaction portion */
#define TR_ABTC_APPL_RESOURCE_LIMITATION        0x0104  /* resource limitation */

#define TR_ABTC_PRIV_UNREC_PKG_TYPE             0x0201  /* unrecognized package type */
#define TR_ABTC_PRIV_INCORRECT_XACT_PORTION     0x0202  /* incorrect transaction portion */
#define TR_ABTC_PRIV_BAD_XACT_PORTION           0x0203  /* badly structured transaction
                                                           portion */
#define TR_ABTC_PRIV_UNASSIGNED_RESP_TRANS_ID   0x0204  /* unassigned responding
                                                           transaction id */
#define TR_ABTC_PRIV_PERM_TO_RELEASE_PROB       0x0205  /* permission to release problem */
#define TR_ABTC_PRIV_RESOURCE_UNAVAIL           0x0206  /* resource unavailable */
#define TR_ABTC_PRIV_UNREC_DIALOG_PORTION_ID    0x0207  /* unrecognized dialogue portion
                                                           id */
#define TR_ABTC_PRIV_BAD_DIALOG_PORTION         0x0208  /* badly structured dialogue
                                                           portion */
```

```
#define TR_ABTC_PRIV_MISSING_DIALOG_PORTION     0x0209  /* missing dialogue portion */
#define TR_ABTC_PRIV_INCONSIST_DIALOG_PORTION   0x020a  /* inconsistent dialogue portion */

/*
 *  TR_INFO_REQ.  This primitive consists of one M_PCPROTO message block.
 */
typedef struct TR_info_req {
    t_uscalar_t PRIM_type;      /* Always TR_INFO_REQ */
} TR_info_req_t;

/*
 *  TR_INFO_ACK.  This primitive consists of one M_PCPROTO message block.
 */
typedef struct TR_info_ack {
    t_scalar_t PRIM_type;       /* Always TR_INFO_ACK */
    t_scalar_t TSDU_size;       /* maximum TSDU size */
    t_scalar_t ETSDU_size;      /* maximum ETSDU size */
    t_scalar_t CDATA_size;      /* connect data size */
    t_scalar_t DDATA_size;      /* discon data size */
    t_scalar_t ADDR_size;       /* address size */
    t_scalar_t OPT_size;        /* options size */
    t_scalar_t TIDU_size;       /* transaction i/f data unit size */
    t_scalar_t SERV_type;       /* service type */
    t_scalar_t CURRENT_state;   /* current state */
    t_scalar_t PROVIDER_flag;   /* type of TR provider */
    t_scalar_t TRPI_version;    /* version # of trpi that is supported */
} TR_info_ack_t;

/*
 *  TR_BIND_REQ.  This primitive consists of one M_PROTO message block.
 */
typedef struct TR_bind_req {
    t_uscalar_t PRIM_type;      /* Always TR_BIND_REQ */
    t_uscalar_t ADDR_length;    /* address length */
    t_uscalar_t ADDR_offset;    /* address offset */
    t_uscalar_t XACT_number;    /* maximum outstanding transaction reqs. */
    t_uscalar_t BIND_flags;     /* bind flags */
} TR_bind_req_t;

/*
 *  TR_BIND_ACK.  This primitive consists of one M_PROTO message block.
 */
typedef struct TR_bind_ack {
    t_uscalar_t PRIM_type;      /* Always TR_BIND_ACK */
    t_uscalar_t ADDR_length;    /* address length */
    t_uscalar_t ADDR_offset;    /* address offset */
    t_uscalar_t XACT_number;    /* open transactions */
    t_uscalar_t TOKEN_value;    /* value of "token" assigned to stream */
} TR_bind_ack_t;

/*
 *  TR_ADDR_REQ.  This primitive consists of one M_PROTO message block.
 */
typedef struct TR_addr_req {
    t_uscalar_t PRIM_type;      /* Always TR_ADDR_REQ */
    t_uscalar_t TRANS_id;       /* Transaction id */
```

```
} TR_addr_req_t;

/*
 *  TR_ADDR_ACK.  This primitive consists of one M_PCPROTO message block.
 */
typedef struct TR_addr_ack {
    t_uscalar_t PRIM_type;      /* Always TR_ADDR_ACK */
    t_uscalar_t LOCADDR_length; /* local address length */
    t_uscalar_t LOCADDR_offset; /* local address offset */
    t_uscalar_t REMADDR_length; /* remote address length */
    t_uscalar_t REMADDR_offset; /* remote address offset */
} TR_addr_ack_t;

/*
 *  TR_UNBIND_REQ.  This primtive consists of one M_PROTO message block.
 */
typedef struct TR_unbind_req {
    t_uscalar_t PRIM_type;      /* Always TR_UNBIND_REQ */
} TR_unbind_req_t;

/*
 *  TR_OPTMGMT_REQ.  This primtive consists of one M_PROTO message block.
 */
typedef struct TR_optmgmt_req {
    t_uscalar_t PRIM_type;      /* Always T_OPTMGMT_REQ */
    t_uscalar_t OPT_length;     /* options length */
    t_uscalar_t OPT_offset;     /* options offset */
    t_uscalar_t MGMT_flags;     /* options data flags */
} TR_optmgmt_req_t;

/*
 *  TR_OPTMGMT_ACK.  This primitive consists of one M_PCPROTO message block.
 */
typedef struct TR_optmgmt_ack {
    t_uscalar_t PRIM_type;      /* Always T_OPTMGMT_ACK */
    t_uscalar_t OPT_length;     /* options length */
    t_uscalar_t OPT_offset;     /* options offset */
    t_uscalar_t MGMT_flags;     /* options data flags */
} TR_optmgmt_ack_t;

/*
 *  TR_OK_ACK.  This primitive consists of one M_PCPROTO message block.
 */
typedef struct TR_ok_ack {
    t_uscalar_t PRIM_type;      /* Always T_OK_ACK */
    t_uscalar_t CORRECT_prim;   /* correct primitive */
} TR_ok_ack_t;

/*
 *  TR_ERROR_ACK.  This primitive consists of one M_PCPROTO message block.
 */
typedef struct TR_error_ack {
    t_uscalar_t PRIM_type;      /* Always T_ERROR_ACK */
    t_uscalar_t ERROR_prim;     /* primitive in error */
    t_uscalar_t TRPI_error;     /* TRPI error code */
    t_uscalar_t UNIX_error;     /* UNIX error code */
```

```
    t_uscalar_t TRANS_id;       /* Transaction id */
} TR_error_ack_t;

/*
 *  TR_UNI_REQ.  This primitive consists of one M_PROTO message block followed
 *  by one or more M_DATA blocks.
 */
typedef struct TR_uni_req {
    t_uscalar_t PRIM_type;      /* Always TR_UNI_REQ */
    t_uscalar_t DEST_length;    /* Destination address length */
    t_uscalar_t DEST_offset;    /* Destination address offset */
    t_uscalar_t ORIG_length;    /* Originating address length */
    t_uscalar_t ORIG_offset;    /* Originating address offset */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_uni_req_t;

/*
 *  TR_UNI_IND.  This primitive consists of one M_PROTO message block followed
 *  by one or more M_DATA blocks.
 */
typedef struct TR_uni_ind {
    t_uscalar_t PRIM_type;      /* Always TR_UNI_REQ */
    t_uscalar_t DEST_length;    /* Destination address length */
    t_uscalar_t DEST_offset;    /* Destination address offset */
    t_uscalar_t ORIG_length;    /* Originating address length */
    t_uscalar_t ORIG_offset;    /* Originating address offset */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_uni_ind_t;

/*
 *  TR_BEGIN_REQ.
 */
typedef struct TR_begin_req {
    t_uscalar_t PRIM_type;      /* Always TR_BEGIN_REQ */
    t_uscalar_t CORR_id;        /* Correlation id */
    t_uscalar_t ASSOC_flags;    /* Association flags */
    t_uscalar_t DEST_length;    /* Destination address length */
    t_uscalar_t DEST_offset;    /* Destination address offset */
    t_uscalar_t ORIG_length;    /* Originating address length */
    t_uscalar_t ORIG_offset;    /* Originating address offset */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_begin_req_t;

/*
 *  TR_BEGIN_IND.
 */
typedef struct TR_begin_ind {
    t_uscalar_t PRIM_type;      /* Always TR_BEGIN_IND */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t ASSOC_flags;    /* Association flags */
    t_uscalar_t DEST_length;    /* Destination address length */
    t_uscalar_t DEST_offset;    /* Destination address offset */
    t_uscalar_t ORIG_length;    /* Originating address length */
```

```
    t_uscalar_t ORIG_offset;    /* Originating address offset */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_begin_ind_t;

/*
 *  TR_BEGIN_RES.
 *
 *  This primitive represents the first TR-CONTINUE response to a TR-BEGIN
 *  indication.
 */
typedef struct TR_begin_res {
    t_uscalar_t PRIM_type;      /* Always TR_BEGIN_RES */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t ASSOC_flags;    /* Association flags */
    t_uscalar_t ORIG_length;    /* Originating address length */
    t_uscalar_t ORIG_offset;    /* Originating address offset */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_begin_res_t;

/*
 *  TR_BEGIN_CON.
 *
 *  This primitive represents the first TR-CONTINUE configuration of a
 *  TR-BEGIN request.
 */
typedef struct TR_begin_con {
    t_uscalar_t PRIM_type;      /* Always TR_BEGIN_CON */
    t_uscalar_t CORR_id;        /* Correlation Id */
    t_uscalar_t ASSOC_flags;    /* Association flags */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t ORIG_length;    /* Originating address length */
    t_uscalar_t ORIG_offset;    /* Originating address offset */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_begin_con_t;

/*
 *  TR_CONT_REQ.
 */
typedef struct TR_cont_req {
    t_uscalar_t PRIM_type;      /* Always TR_CONT_REQ */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t ASSOC_flags;    /* Association flags */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_cont_req_t;

/*
 *  TR_CONT_IND.
 */
typedef struct TR_cont_ind {
    t_uscalar_t PRIM_type;      /* Always TR_CONT_IND */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t ASSOC_flags;    /* Association flags */
```

```
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_cont_ind_t;

/*
 *  TR_END_REQ.
 */
typedef struct TR_end_req {
    t_uscalar_t PRIM_type;      /* Always TR_END_REQ */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t TERM_scenario;  /* Termination scenario */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_end_req_t;

/*
 *  TR_END_IND.
 */
typedef struct TR_end_ind {
    t_uscalar_t PRIM_type;      /* Always TR_END_IND */
    t_uscalar_t CORR_id;        /* Correlation id */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_end_ind_t;

/*
 *  TR_ABORT_REQ.
 */
typedef struct TR_abort_req {
    t_uscalar_t PRIM_type;      /* Always TR_ABORT_REQ */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t ABORT_cause;    /* Cause of the abort */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
} TR_abort_req_t;

/*
 *  TR_ABORT_IND.
 */
typedef struct TR_abort_ind {
    t_uscalar_t PRIM_type;      /* Always TR_ABORT_IND */
    t_uscalar_t CORR_id;        /* Correlation id */
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t OPT_length;     /* Options structure length */
    t_uscalar_t OPT_offset;     /* Options structure offset */
    t_uscalar_t ABORT_cause;    /* Cause of the abort */
    t_uscalar_t ORIGINATOR;     /* Originator P or U */
} TR_abort_ind_t;

/*
 *  TR_NOTICE_IND.
 */
typedef struct TR_notice_ind {
    t_uscalar_t PRIM_type;      /* Always TR_NOTICE_IND */
    t_uscalar_t CORR_id;        /* Correlation id */
```

```
    t_uscalar_t TRANS_id;       /* Transaction id */
    t_uscalar_t REPORT_cause;   /* SCCP return cause */
} TR_notice_ind_t;
```

License

GNU Free Documentation License

GNU FREE DOCUMENTATION LICENSE
Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Terms and Conditions for Copying, Distribution and Modification

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a

textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

   B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

   C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

   D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgments" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be

added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgments", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

END OF TERMS AND CONDITIONS

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being list their titles, with the
Front-Cover Texts being list, and with the Back-Cover Texts being list.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being *list*"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Glossary

Signalling Data Link Service Data Unit
: A grouping of SDL user data whose boundaries are preserved from one end of the signalling data link connection to the other.

Data transfer
: The phase in connection and connectionless modes that supports the transfer of data between to signalling data link users.

SDL provider
: The signalling data link layer protocol that provides the services of the signalling data link interface.

SDL user
: The user-level application or user-level or kernel-level protocol that accesses the services of the signalling data link layer.

Local management
: The phase in connection and connectionless modes in which a SDL user initializes a *Stream* and attaches a PPA address to the *Stream*. Primitives in this phase generate local operations only.

PPA
: The point at which a system attaches itself to a physical communications medium.

PPA identifier
: An identifier of a particular physical medium over which communication transpires.

Acronyms

ITU-T	International Telecommunications Union - Telecom Sector
PPA	Physical Point of Attachment
SDLI	Signalling Data Link Interface
SDL SDU	Signalling Data Link Service Data Unit
SDL	Signalling Data Link

References

1. ITU-T Recommendation X.210, (Geneva, 1993), "Information Technology — Open Systems Interconnection — Basic reference model: Conventions for the definition of OSI services," ISO/IEC 10731:1994.
2. ITU-T Recommendation X.217, (Geneva, 1995), "Information Technology — Open Systems Interconnection — Service definition for the Association Control Service Element," ISO/IEC 8649:1996.
3. ITU-T Recommendation X.227, (Geneva, 1995), "Information Technology — Open Systems Interconnection — Connection-oriented protocol for the Association Control Service Element: Protocol Specification," ISO/IEC 8650-1.
4. ITU-T Recommendation X.237, (Geneva, 1995), "Information Technology — Open Systems Interconnection — Connectionless protocol for the Association Control Service Element: Protocol Specification," ISO/IEC 10035-1 : 1995.
5. ITU-T Recommendation X.216, (Geneva, 1994), "Information Technology — Open Systems Interconnection — Presentation service definition," ISO/IEC 8822:1994.
6. ITU-T Recommendation X.226, (Geneva, 1994), "Information Technology — Open Systems Interconnection — Connection-oriented presentation protocol: Protocol specification," ISO/IEC 8823-1:1994.
7. ITU-T Recommendation X.236, (Geneva, 1995), "Information Technology — Open Systems Interconnection — Connectionless presentation protocol: Protocol specification," ISO/IEC 9576-1:1995.
8. ITU-T Recommendation X.215, (Geneva, 1995), "Information Technology — Open Systems Interconnection — Session service definition," ISO/IEC 8326:1996.
9. ITU-T Recommendation X.225, (Geneva, 1995), "Information Technology — Open Systems Interconnection — Connection-oriented session protocol: Protocol specification," ISO/IEC 8327-1:1996.
10. ITU-T Recommendation X.235, (Geneva, 1995), "Information Technology — Open Systems Interconnection — Connectionless session protocol: Protocol specification," ISO/IEC 9548-1:1995.
11. ITU-T Recommendation X.214, (Geneva, 1995), "Information Technology — Open Systems Interconnection — Transport service definition," ISO/IEC 8072:1996.
12. ITU-T Recommendation X.224
13. ITU-T Recommendation Q.700
14. ITU-T Recommendation Q.701
15. ITU-T Recommendation Q.702
16. ITU-T Recommendation Q.703
17. ITU-T Recommendation Q.704
18. Geoffrey Gerrien, "CDI - Application Program Interface Guide," Gcom, Inc., March 1999.
19. ITU-T Recommendation Q.771, (Geneva, 1993), "Signalling System No. 7 — Functional description of transaction capabilities," (White Book).

Index

T

U

X