Transport Provider Interface Specification

UNIX International
OSI Special Interest Group
Revision: 1.5 (Date: 92/12/10)

Published by:

UNIX International
Waterview Corporate Center
20 Waterview Boulevard
Parsippany, NJ 07054

for further information, contact:
Vice President of Marketing

Phone: +1 201-263-8400
Fax: +1 201-263-8401

International Offices:

UNIX International Asian/Pacific Office Shinei Bldg. 1F Kameido Koto-ku, Tokyo 136 Japan	UNIX International Australian Office Suite 1, Century Plaza 80 Berry Street North Sydney, NSW 2060 Australia	UNIX International European Office 25, Avenue de Beaulieu 1160 Brussels Belgium	UNIX International Pacific Basin Office Cintech II 75 Science Park Drive Singapore Science Park Singapore 0511 Singapore
Phone:(81) 3-3636-1122 Fax: (81) 3-3636-1121	Phone:(61) 2-922-5341 Fax: (61) 2-929-0635	Phone:(32) 2-672-3700 Fax: (32) 2-672-4415	Phone:(65) 776-0313 Fax: (65) 776-0421

NOTICE:

This document is based on the UNIX System Laboratories Transport Provider Interface (TPI) specification which was used with permission by the UNIX International OSI Special Interest Group (UI OSISIG). Participation in the UI OSISIG is open to UNIX International members and other interested parties. For further information contact UNIX International at the addresses above.

UNIX International is making this documentation available as a reference point for the industry. While UNIX International believes that these interfaces are well defined in this release of the document, minor changes may be made prior to products conforming to the interfaces being made available from UNIX System Laboratories or UNIX International members.

Trademarks:

1. Introduction

To support a framework for providing networking products in the UNIX system, an effort is underway to define service interfaces that map to strategic levels of the Open Systems Interconnection (OSI) Reference Model. These service interfaces hide implementation details of a particular service from the consumer of the service. This enables system programmers to develop software independent of the particular protocol that provides a specific service. The interfaces being specified for UNIX System V are defined within the STREAMS environment. This document specifies a kernel-level interface that supports the services of the Transport Layer for connection-mode and connectionless-mode services.

This specification applies to System V Release 4.2 ES/MP.

2. Transport Provider Interface

The transport interface defines a message interface to a transport provider implemented under STREAMS[1]. This version of the transport provider interface supports the XPG4 version of the X/Open Transport Interface (XTI). A user communicates to a transport provider via a full duplex path known as a *stream* (see figure 1). This *stream* provides a mechanism in which messages may be passed to the transport provider from the transport user and vice versa.

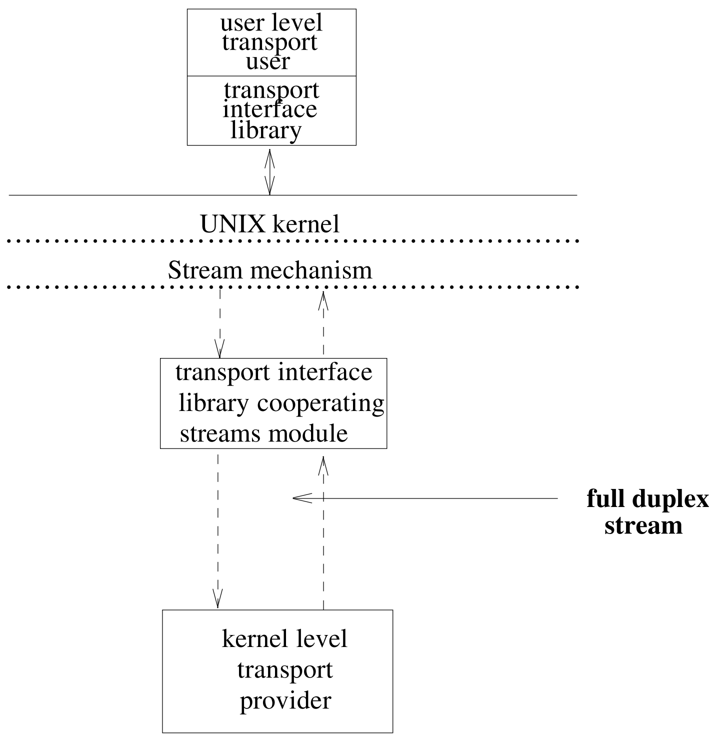

Figure 1. Example of a *stream* from a user to a transport provider

The STREAMS messages that are used to communicate transport service primitives between the transport user and the transport 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 transport service primitive and all the relevant arguments associated with the primitive. The M_DATA blocks contain transport user data associated with the transport service primitive.

1. It is assumed that the reader of this document is familiar with the concept STREAMS.

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

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

For both types of transport service, two types of primitives exist: primitives which originate from the transport user and primitives which originate from the transport provider. The primitives which originate from the transport user make requests to the transport provider or respond to an event of the transport provider. The primitives which originate from the transport provider are either confirmations of a request or are indications to the transport user that an event has occurred. Section 2 lists the primitive types along with the mapping of those primitives to the STREAMS message types and the transport primitives of the ISO IS 8072 and IS 8072/DAD transport service definitions. The format of these primitives and the rules governing the use of them are described in sections 2.1, 2.2, and 2.3.

2.1 Common Transport Primitives

The following transport primitives are common to both the connection-mode and connectionless-mode transport services.

2.1.1 User-Originated Primitives

The following describes the format of the transport primitives which are generated by the transport user.

2.1.1.1 T_INFO_REQ - get transport protocol parameter sizes.

This primitive requests the transport provider to return the sizes of all relevant protocol parameters, plus the current state of the provider[2]. The format of the message is one M_PCPROTO message block. The format of the M_PCPROTO message block is as follows:

```
struct T_info_req {
        long PRIM_type;             /* always T_INFO_REQ */
}
```

Where *PRIM_type* indicates the primitive type.

This primitive requires the transport provider to generate one of the following acknowledgments upon receipt of the primitive and that the transport user wait for the acknowledgment prior to issuing any other primitives:

- Successful

  Acknowledgment of the primitive via the T_INFO_ACK described in section 2.1.2.1.

- Non-fatal errors

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

2.1.1.2 T_BIND_REQ - bind protocol address request.

This primitive requests that the transport provider bind a protocol address to the *stream*, negotiate the number of connect indications allowed to be outstanding by the transport provider for the specified protocol address, and activate[3] the *stream* associated with the protocol address. The format of the message is one M_PROTO message block. The format of the M_PROTO message block is as follows:

---

2. The T_INFO_REQ and T_INFO_ACK primitives have no effect on the state of the transport provider and do not appear in the state tables.

3. A *stream* is viewed as active when the transport provider may receive and transmit TPDUs (transport protocol data units) associated with the *stream*.

```
struct T_bind_req {
        long PRIM_type;                  /* always T_BIND_REQ */
        long ADDR_length;           /* length of address */
        long ADDR_offset;           /* offset of address */
        unsigned long CONIND_number;     /* requested number of connect indications
                                        to be queued */
}
```

Where *PRIM_type* indicates the primitive type. *ADDR_length* is the length[4] of the protocol address to be bound to the *stream* and *ADDR_offset* is the offset from the beginning of the M_PROTO block where the protocol address begins. *CONIND_number*[5] is the requested number of connect indications[6] allowed to be outstanding by the transport provider for the specified protocol address. 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 transport user. For rules governing the requests made by this primitive see the T_BIND_ACK primitive in section 2.1.2.2.

This primitive requires the transport provider to generate one of the following acknowledgments upon receipt of the primitive, and the transport user must wait for the acknowledgment before issuing any other primitives:

- Successful

  Correct acknowledgment of the primitive is indicated via the T_BIND_ACK primitive.

- Non-fatal errors

  These errors will be indicated via the T_ERROR_ACK primitive described in section 2.1.2.4. The allowable errors are as follows:

  TBADADDR — This indicates that the protocol address was in an incorrect format or the address contained illegal information. It is not intended to indicate protocol errors.

  TNOADDR — This indicates that the transport provider could not allocate an address.

---

4. All lengths, offsets, and sizes in all structures refer to the number of bytes.
5. This field should be ignored by those providing a connectionless transport service.
6. If the number of outstanding connect indications equals CONIND_number, the transport provider need not discard further incoming connect indications, but may chose to queue them internally until the number of outstanding connect indications drops below CONIND_number.

TACCES	This indicates that the user did not have proper permissions for the use of the requested address.
TOUTSTATE	The primitive would place the transport interface out of state.
TSYSERR	A system error has occurred and the UNIX System error is indicated in the primitive.
TADDRBUSY	This indicates that the requested address is already in use.

2.1.1.3 T_UNBIND_REQ - unbind protocol address request.

This primitive requests that the transport provider unbind the protocol address associated with the *stream* and deactivate the *stream*. The format of the message is one M_PROTO message block. The format of the M_PROTO message block is as follows:

```
struct T_unbind_req {
        long PRIM_type;                /* always T_UNBIND_REQ */
}
```

Where *PRIM_type* indicates the primitive type.

This primitive requires the transport provider to generate the following acknowledgments upon receipt of the primitive and that the transport user must wait for the acknowledgment before issuing any other primitives:

- Successful

Correct acknowledgment of the primitive is indicated via the T_OK_ACK primitive described in section 2.1.2.5.

- Non-fatal errors

These errors will be indicated via the T_ERROR_ACK primitive described in section 2.1.2.4. The allowable errors are as follows:

TOUTSTATE	The primitive would place the transport interface out of state.
TSYSERR	A system error has occurred and the UNIX System error is indicated in the primitive.

2.1.1.4 T_OPTMGMT_REQ - options management.

This primitive allows the transport user to manage the options associated with the *stream*. The format of the message is one M_PROTO message block. The format of the M_PROTO message block is as follows:

```
struct T_optmgmt_req {
        long PRIM_type;             /* always T_OPTMGMT_REQ */
        long OPT_length;            /* options length */
        long OPT_offset;            /* options offset */
        long MGMT_flags;    /* flags */
}
```

Where *PRIM_type* indicates the primitive type. *OPT_length* is the length of the protocol options associated with the primitive and *OPT_offset* is the offset from the beginning of the M_PROTO block where the options begin. The proper alignment of the options is not guaranteed. The options are however, aligned the same as it was received from the transport user. *MGMT_flags* are the flags which define the request made by the transport user. The allowable flags are:

T_NEGOTIATE Negotiate and set the options with the transport provider.

T_CHECK Check the validity of the specified options.

T_CURRENT Return the options currently in effect.

T_DEFAULT Return the default options.

For the rules governing the requests made by this primitive see the T_OPTMGMT_ACK primitive in section 2.1.2.3.

This primitive requires the transport provider to generate one of the following acknowledgments upon receipt of the primitive and that the transport user wait for the acknowledgment before issuing any other primitives:

- Successful

  Acknowledgment of the primitive via the T_OPTMGMT_ACK.

- Non-fatal errors

  These errors will be indicated via the T_ERROR_ACK primitive described in section 2.1.2.4. The allowable errors are as follows:

  TACCES This indicates that the user did not have proper permissions for the use of the requested options.

  TOUTSTATE The primitive would place the transport interface out of state.

  TBADOPT This indicates that the options as specified were in an incorrect format, or they contained illegal information.

  TBADFLAG This indicates that the flags as specified were incorrect or illegal.

  TSYSERR A system error has occurred and the UNIX System error is indicated in the primitive.

TNOTSUPPORT This transport provider does not support the requested flag (T_CHECK or T_CURRENT).

2.1.1.5 T_ADDR_REQ - get protocol addresses request.

This primitive requests that the transport provider return the local protocol address that is bound to the *stream* and the address of the remote transport entity if a connection has been established. The format of the message is one M_PROTO message block. The format of the M_PROTO message block is as follows:

```
struct T_addr_req {
        long PRIM_type;                 /* always T_ADDR_REQ */
}
```

Where *PRIM_type* indicates the primitive type.

This primitive requires the transport provider to generate one of the following acknowledgments upon receipt of the primitive, and the transport user must wait for the acknowledgment before issuing any other primitives:

- Successful

  Correct acknowledgment of the primitive is indicated via the T_ADDR_ACK primitive.

- Non-fatal errors

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

2.1.2 Provider-Originated Primitives

The following describes the format of the transport primitives which are generated by the transport provider.

2.1.2.1 T_INFO_ACK - protocol information acknowledgment.

This primitive indicates to the transport user any relevant protocol-dependent parameters. It should be initiated in response to the T_INFO_REQ primitive described above. The format of this message is one M_PCPROTO message block. The format of the M_PCPROTO message block is as follows:

```
struct T_info_ack {
        long PRIM_type;              /* always T_INFO_ACK */
        long TSDU_size;              /* max TSDU size */
        long ETSDU_size;             /* max ETSDU size */
        long CDATA_size;             /* Connect data size */
        long DDATA_size;             /* Discon data size */
        long ADDR_size;              /* TSAP size */
        long OPT_size;               /* options size */
        long TIDU_size;              /* TIDU size */
        long SERV_type;              /* service type */
        long CURRENT_state;          /* current state */
        long PROVIDER_flag;          /* provider flags */
}
```

where the fields of this message have the following meanings:

PRIM_type This indicates the primitive type.

TSDU_size A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of a TSDU; and a value of -2 specifies that the transfer of normal data is not supported by the transport provider.

ETSDU_size A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of an ETSDU; and a value of -2 specifies that the transfer of expedited data is not supported by the transport provider.

CDATA_size A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment primitives; and a value of -2 specifies that the transport provider does not allow data to be sent with connection establishment primitives.

DDATA_size A value greater than or equal to zero specifies the maximum amount of data that may be associated with the disconnect primitives; and a value of -2 specifies that the transport provider does not allow data to be sent with the disconnect primitives.

ADDR_size A value greater than or equal to zero indicates the maximum size of a transport protocol address; and a value of -2 specifies that the transport provider does not provide user access to transport protocol addresses.

OPT_size — A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of -2 specifies that the transport provider does not support user-settable options although they're read-only; and a value of -3 specifies that the transport provider does not support any options.

TIDU_size[7] — This is the size of the transport protocol interface data unit, and should not exceed the tunable system limit, if non-zero, for the size of a STREAMS message.

SERV_type — This field specifies the service type supported by the transport provider, and is one of the following:

- T_COTS — The provider service is connection oriented with no orderly release support.
- T_COTS_ORD — The provider service is connection oriented with orderly release support.
- T_CLTS — The provider service is a connectionless transport service.

CURRENT_state — This is the current state of the transport provider.

PROVIDER_flag — This field specifies additional properties specific to the transport provider and may alter the way the transport user communicates. Transport providers supporting the features of XTI in XPG4 and beyond must send up a version number as specified below. The following flags may be set by the provider:

- SENDZERO — This flag indicates that the transport provider supports the sending of zero-length TSDUs.
- XPG4_1 — This indicates that the transport provider conforms to XTI in XPG4 and supports the new primitives T_ADDR_REQ and T_ADDR_ACK.

The following rules apply when the type of service is T_CLTS:

— The ETSDU_size, CDATA_size and DDATA_size fields should be -2.

— The TSDU_size should equal the TIDU_size.

7. This is the amount of user data that may be present in a single T_DATA_REQ or T_EXDATA_REQ primitive.

2.1.2.2 T_BIND_ACK - bind protocol address acknowledgment.

This primitive indicates to the transport user that the specified protocol address has been bound to the *stream*, that the specified number of connect indications are allowed to be queued by the transport provider for the specified protocol address, and that the *stream* associated with the specified protocol address has been activated. The format of the message is one M_PCPROTO message block. The format of the M_PCPROTO message block is as follows:

```
struct T_bind_ack {
        long PRIM_type;                   /* always T_BIND_ACK */
        long ADDR_length;         /* length of address - see note in sec. 1.4 */
        long ADDR_offset;         /* offset of address */
        unsigned long CONIND_number;      /* connect indications to be queued */
}
```

Where *PRIM_type* indicates the primitive type. *ADDR_length* is the length of the protocol address that was bound to the *stream* and *ADDR_offset* is the offset from the beginning of the M_PCPROTO block where the protocol address begins. *CONIND_number*[8] is the accepted number of connect indications allowed to be outstanding by the transport provider for the specified protocol address. The proper alignment of the address in the M_PCPROTO message block is not guaranteed.

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

— If the ADDR_length field in the T_BIND_REQ primitive is 0, then the transport provider must assign a transport protocol address to the user.

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

The following rules apply to negotiating the *CONIND_number* argument:

— The returned value must be less than or equal to the corresponding requested number as indicated in the T_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 may have a negotiated accepted number of maximum connect requests greater than zero. If a T_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 transport provider must return an error.

---

8. This field doesn't apply to connectionless transport providers.

— If a *stream* with CONIND_number greater than zero is used to accept a connection, 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 CONIND_number greater than zero. This will prevent more than one *stream* bound to the identical protocol address from accepting connect indications.

— A *stream* requesting a *CONIND_number* of zero should always be legal. This indicates to the transport provider that the *stream* is to be used to request connections only.

— A *stream* with a negotiated *CONIND_number* greater than zero may generate connect requests or accept connect indications.

If the above rules result in an error condition, then the transport provider must issue an T_ERROR_ACK primitive to the transport user specifying the error as defined in the description of the T_BIND_REQ primitive.

2.1.2.3 T_OPTMGMT_ACK - option management acknowledgment.

This indicates to the transport user that the options management request has completed. The format of the message is one M_PCPROTO message block. The format of the M_PCPROTO message block is as follows:

```
struct T_optmgmt_ack {
        long PRIM_type;            /* always T_OPTMGMT_ACK */
        long OPT_length;           /* options length - see note in sec. 1.4 */
        long OPT_offset;           /* options offset */
        long MGMT_flags;     /* flags */
}
```

Where *PRIM_type* indicates the primitive type. *OPT_length* is the length of the protocol options associated with the primitive and *OPT_offset* is the offset from the beginning of the M_PCPROTO block where the options begin. The proper alignment of the options is not guaranteed. *MGMT_flags* should be the same as those specified in the T_OPTMGMT_REQ primitive with any additional flags as specified below.

The following rules apply to the T_OPTMGMT_ACK primitive.

— If the value of MGMT_flags in the T_OPTMGMT_REQ primitive is T_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 T_OPTMGMT_REQ primitive is T_CHECK, the provider should return the options as specified in the T_OPTMGMT_REQ primitive along with the additional flags T_SUCCESS or T_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 T_OPTMGMT_REQ primitive is T_NEGOTIATE, the provider should set and negotiate the option as specified by the following rules:

— If the OPT_length field of the T_OPTMGMT_REQ primitive is 0, then the transport provider is to set and return the default options associated with the *stream* in the T_OPTMGMT_ACK primitive.

— If options are specified in the T_OPTMGMT_REQ primitive, then the transport provider should negotiate those options, set the negotiated options and return the negotiated options in the T_OPTMGMT_ACK primitive. It is the user's responsibility to check the negotiated options returned in the T_OPTMGMT_ACK primitive and take appropriate action.

— If the value of MGMT_flags in the T_OPTMGMT_REQ primitive is T_CURRENT, the provider should return the current options that are currently associated with the *stream*.

— If the value of MGMT_flags in the T_OPTMGMT_REQ primitive is either T_NEGOTIATE or T_CHECK and the transport provider cannot support the requested flag, an error is to be returned.

If the above rules result in an error condition, the transport provider must issue a T_ERROR_ACK primitive to the transport user specifying the error as defined in the description of the T_OPTMGMT_REQ primitive.

2.1.2.4 T_ERROR_ACK - error acknowledgment.

This primitive indicates to the transport user that a non-fatal[9] error has occurred in the last transport-user-originated primitive. This may only be initiated as an acknowledgment for those primitives that require one. It also indicates to the user that no action was taken on the primitive that caused the error. The format of the message is one M_PCPROTO message block. The format of the M_PCPROTO message block is as follows:

```
struct T_error_ack {
        long PRIM_type;         /* always T_ERROR_ACK */
        long ERROR_prim;        /* primitive in error */
        long TLI_error;         /* TLI error code - see note in sec. 1.4 */
        long UNIX_error;        /* UNIX error code - see note in sec. 1.4 */
}
```

Where *PRIM_type* identifies the primitive. *ERROR_prim* identifies the primitive type that caused the error and *TLI_error* contains the Transport Level Interface error code. *UNIX_error* contains the UNIX System error code. This may only be non zero if *TLI_error* is equal to TSYSERR. The following Transport Level Interface error codes are allowed to be returned:

9. For a overview of the error handling capabilities available to the transport provider see section 2.4.

TBADADDR	This indicates that the protocol address as specified in the primitive was in an incorrect format or the address contained illegal information.
TBADOPT	This indicates that the options as specified in the primitive were in an incorrect format, or they contained illegal information.
TBADF	This indicates that the *stream* queue pointer as specified in the primitive was illegal.
TNOADDR	This indicates that the transport provider could not allocate an address.
TACCES	This indicates that the user did not have proper permissions.
TOUTSTATE	The primitive would place the interface out of state.
TBADSEQ	The sequence number specified in the primitive was incorrect or illegal.
TBADFLAG	The flags specified in the primitive were incorrect or illegal.
TBADDATA	The amount of user data specified was illegal.
TSYSERR	A system error has occurred and the UNIX System error is indicated in the primitive.
TADDRBUSY	The requested address is in use.
TRESADDR	The transport provider requires that the responding stream is bound to the same address as the stream on which the connection indication was received.
TNOTSUPPORT	The transport provider does not support the requested capability.

2.1.2.5 T_OK_ACK - success acknowledgment.

This primitive indicates to the transport user that the previous transport-user-originated primitive was received successfully by the transport provider. It does not indicate to the transport user any transport protocol action taken due to the issuance of the last primitive. This may only be initiated as an acknowledgment for those primitives that require one. The format of the message is one M_PCPROTO message block. The format of the M_PCPROTO message block is as follows:

```
struct T_ok_ack {
        long PRIM_type;                 /* always T_OK_ACK */
        long CORRECT_prim;/* primitive */
}
```

Where *PRIM_type* identifies the primitive. *CORRECT_prim* contains the successfully received primitive type.

2.1.2.6 T_ADDR_ACK - get protocol addresses acknowledgment.

This primitive indicates to the transport user the addresses of the local and remote transport entities. The local address is the protocol address that has been bound to the *stream*. If a connection has been established, the remote address is the protocol address of the remote transport entity. The format of the message is one M_PCPROTO message block. The format of the M_PCPROTO message block is as follows:

```
struct T_addr_ack {
        long PRIM_type;               /* always T_ADDR_ACK */
        long LOCADDR_length;          /* length of local address - see note in sec. 1.4 */
        long LOCADDR_offset;          /* offset of local address */
        long REMADDR_length;          /* length of remote address - see note in sec. 1.4 */
        long REMADDR_offset;          /* offset of remote address */
}
```

Where *PRIM_type* indicates the primitive type. *LOCADDR_length* is the length of the protocol address that was bound to the *stream* and *LOCADDR_offset* is the offset from the beginning of the M_PCPROTO block where the protocol address begins. If the stream is in the data transfer state, *REMADDR_length* is the length of the protocol address of the remote transport entity and *REMADDR_offset* is the offset from the beginning of the M_PCPROTO block where the protocol address begins.

The following rules apply:

— If the interface is in any state but T_DATAXFER, the values returned for REMADDR_length and REMADDR_offset must be 0.

— If the interface is in the T_UNINIT or T_UNBND state, the values returned for LOCADDR_length and LOCADDR_offset must be 0.

2.2 Connection-Mode Transport Primitives

The following transport primitives pertain only to the connection-mode transport service.

2.2.1 User-Originated Primitives

The following describes the format of the transport primitives which are generated by the transport user.

2.2.1.1 T_CONN_REQ - connect request.

This primitive requests that the transport provider make a connection to the specified destination. The format of this message is one M_PROTO message block followed by zero or more M_DATA blocks if any user data is specified by the transport user. The format of the M_PROTO message block is as follows:

```
struct T_conn_req {
        long    PRIM_type;    /* always T_CONN_REQ */
        long    DEST_length;  /* dest addr length */
        long    DEST_offset;  /* dest addr offset */
        long    OPT_length;   /* options length */
        long    OPT_offset;   /* options offset */
}
```

Where *PRIM_type* identifies the primitive type. *DEST_length* is the length of the destination address and *DEST_offset* is the offset from the beginning of the M_PROTO message block where the destination address begins. Similarly, *OPT_length* and *OPT_offset* describe the location of the requested options associated with the primitive. The proper alignment of the destination address and options in the M_PROTO message block is not guaranteed[10]. The destination address and options in the M_PROTO message block are however, aligned the same as they were received from the transport user.

This primitive requires the transport provider to generate one of the following acknowledgments upon receipt of the primitive, and the transport user must wait for the acknowledgment before issuing any other primitives:

- Successful

  Correct acknowledgment of the primitive is indicated via the T_OK_ACK primitive described in section 2.1.2.5.

- Non-fatal errors

---

10. The information located by the defined structures may not be in the proper alignment in the message blocks, so the casting of structure definitions over these fields may produce incorrect results. It is advised that the transport providers supply exact format specifications for the appropriate information to the transport users.

These errors will be indicated via the T_ERROR_ACK primitive described in section 2.1.2.4. The allowable errors are as follows:

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

TBADADDR — 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 error types are indicated via the T_DISCON_IND primitive.

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

TOUTSTATE — The primitive would place the transport interface out of state.

TBADDATA — The amount of user data specified was illegal.

TSYSERR — A system error has occurred and the UNIX System error is indicated in the primitive.

TADDRBUSY — This transport provider does not support multiple connections with the same local and remote addresses.

2.2.1.2 T_CONN_RES - connection response.

This primitive requests that the transport provider accept a previous connect request on the specified response queue. The format of this message is one M_PROTO message block followed by zero or more M_DATA blocks if any user data is specified by the transport user. The format of the M_PROTO message block is as follows:

```
struct T_conn_res {
        long    PRIM_type;    /* always T_CONN_RES */
        queue_t *QUEUE_ptr;/* response queue ptr */
        long    OPT_length;   /* options length */
        long    OPT_offset;   /* options offset */
        long    SEQ_number;  /* sequence number */
}
```

Where *PRIM_type* identifies the primitive type. *QUEUE_ptr* identifies the transport provider queue pair (i.e. read queue pointer) which should be used to accept the connect request. This queue pointer should map onto a *stream* which is already bound to a protocol address but if the *stream* is not bound, the transport provider must bind it to the same protocol address that was bound to the *stream* on which the connection indication arrived. *OPT_length* is the length of the responding options and *OPT_offset* is the offset from the beginning of the M_PROTO message block where the responding options begin.

SEQ_number is the sequence number which identifies the connection to be responded to. The proper alignment of the options in the M_PROTO message block is not guaranteed. The options in the M_PROTO message block are, however, aligned the same as they were received from the transport user.

This primitive requires the transport provider to generate one of the following acknowledgments upon receipt of the primitive, and the transport user wait for the acknowledgment before issuing any other primitives:

- Successful

  Correct acknowledgment of the primitive is indicated via the T_OK_ACK primitive described in section 2.1.2.5.

- Non-fatal errors

  These errors will be indicated via the T_ERROR_ACK primitive described in section 2.1.2.4. The allowable errors are as follows:

TBADF	This indicates that the response queue pointer was illegal.
TBADOPT	This indicates that the options were in an incorrect format, or they contained illegal information.
TACCES	This indicates that the user did not have proper permissions for the use of the options or response id.
TOUTSTATE	The primitive would place the transport interface out of state.
TBADDATA	The amount of user data specified was illegal.
TBADSEQ	The sequence number specified in the primitive was incorrect or illegal.
TSYSERR	A system error has occurred and the UNIX System error is indicated in the primitive.
TRESADDR	The transport provider requires that the responding *stream* is bound to the same address as the *stream* on which the connection indication was received.
TBADADDR	This indicates that the protocol address was in an incorrect format or the address contained illegal information.

2.2.1.3 T_DISCON_REQ - disconnect request.

This primitive requests that the transport provider deny a request for connection, or disconnect an existing connection. The format of this message is one M_PROTO message block possibly followed by one or more M_DATA message blocks if there is any user data specified by the transport user. The format of the M_PROTO message block is as follows:

```
struct T_discon_req {
        long PRIM_type;              /* always T_DISCON_REQ */
        long SEQ_number;     /* sequence number */
}
```

Where *PRIM_type* identifies the primitive type. *SEQ_*number identifies the outstanding connect indication that is to by denied. If the disconnect request is disconnecting an already existing connection, then the value of *SEQ_number* will be ignored.

This primitive requires the transport provider to generate the following acknowledgment upon receipt of the primitive, and the transport user must wait for the acknowledgment prior to issuing any other primitives:

- Successful

  Correct acknowledgment of the primitive is indicated via the T_OK_ACK primitive described in section 2.1.2.5.

- Non-fatal errors

  These errors will be indicated via the T_ERROR_ACK primitive described in section 2.1.2.4. The allowable errors are as follows:

TOUTSTATE	The primitive would place the transport interface out of state.
TBADDATA	The amount of user data specified was illegal.
TBADSEQ	The sequence number specified in the primitive was incorrect or illegal.
TSYSERR	A system error has occurred and the UNIX System error is indicated in the primitive.

2.2.1.4 T_DATA_REQ - data request.

This primitive indicates to the transport provider that this message contains a transport interface data unit. One or more transport interface data units form a transport service data unit (TSDU)[11]. This primitive has a mechanism which indicates the beginning and end of a transport service data unit. However, not all transport providers support the concept of a transport service data unit, as noted in section 2.1.2.1. The format of the message is one M_PROTO message block followed by zero or more M_DATA message blocks where each M_DATA message block contains zero or more bytes of data. The format of the M_PROTO message block is as follows:

---

11. The maximum transport service data unit size allowed by the transport provider is indicated to the transport user via the T_INFO_ACK primitive.

```
struct T_data_req {
        long PRIM_type;              /* always T_DATA_REQ */
        long MORE_flag;              /* indicates more data in TSDU */
}
```

Where *PRIM_type* identifies the primitive type. *MORE_flag* when greater than zero, indicates that the next T_DATA_REQ primitive is also part of this transport service data unit.

The transport provider must also recognize a message of one or more M_DATA message blocks without the leading M_PROTO message block as a T_DATA_REQ primitive. This message type will be initiated from the WRITE(BA_OS) operating system service routine. In this case there are no implied transport service data unit boundaries, and the transport provider may view this message type as a self contained transport service data unit. If these two types of messages are intermixed, then transport service data boundaries may be lost.

This primitive does not require any acknowledgments, although it may generate a fatal error. This is indicated via a M_ERROR message type which results in the failure of all operating system service routines on the *stream*. The allowable errors are as follows:

EPROTO This indicates one of the following unrecoverable protocol conditions:

- The transport service interface was found to be in an incorrect state. If the interface is in the T_IDLE state when the provider receives the T_DATA_REQ primitive, then the transport provider should just drop the message without generating a fatal error.
- The amount of transport user data associated with the primitive defines a transport service data unit larger than that allowed by the transport provider.

2.2.1.5 T_EXDATA_REQ - expedited data request.

This primitive indicates to the transport provider that this message contains an expedited transport interface data unit. One or more expedited transport interface data units form an expedited transport service data unit[12]. This primitive has a mechanism which indicates the beginning and end of an expedited transport service data unit. However, not all transport providers support the concept of an expedited transport service data unit, as noted in section 2.1.2.1. The format of the message is one M_PROTO message block followed by one or more M_DATA message blocks containing at least one byte of data. The format of the M_PROTO message block is as follows:

---

12. The maximum size of a expedited transport service data unit is indicated to the transport user via the T_INFO_ACK primitive.

```
struct T_exdata_req {
        long PRIM_type;              /* always T_EXDATA_REQ */
        long MORE_flag;              /* indicates more data in ETSDU */
}
```

Where *PRIM_type* identifies the primitive type. *MORE_flag* when greater than zero indicates that the next T_EXDATA_REQ primitive is also part of this expedited transport service data unit.

This primitive does not require any acknowledgments, although it may generate a fatal error. This is indicated via a M_ERROR message type which results in the failure of all operating system service routines on the *stream*. The allowable errors are as follows:

EPROTO This indicates one of the following unrecoverable protocol conditions:

- — The transport service interface was found to be in an incorrect state. If the interface is in the T_IDLE state when the provider receives the T_EXDATA_REQ primitive, then the transport provider should just drop the message without generating a fatal error.
- — The amount of transport user data associated with the primitive defines an expedited transport service data unit larger than that allowed by the transport provider.

2.2.1.6 T_ORDREL_REQ - orderly release request.

This primitive indicates to the transport provider that the user is finished sending data. This primitive is only supported by the transport provider if it is of type T_COTS_ORD. The format of the message is one M_PROTO message block. The format of the M_PROTO message block is as follows:

```
struct T_ordrel_req {
        long PRIM_type;              /* always T_ORDREL_REQ */
}
```

Where *PRIM_type* identifies the primitive type.

This primitive does not require any acknowledgments, although it may generate a fatal error. This is indicated via a M_ERROR message type which results in the failure of all operating system service routines on the *stream*. The allowable errors are as follows:

EPROTO This indicates one of the following unrecoverable protocol conditions:

- — The primitive would place the interface in an incorrect state.

2.2.2 Provider-Originated Primitives

The following describes the format of the transport primitives which are generated by the transport provider.

2.2.2.1 T_CONN_IND - connect indication.

This primitive indicates to the transport user that a connect request to the user has been made by the user at the specified source address. The format of this message is one M_PROTO message block followed by zero or more M_DATA blocks if any user data is associated with the primitive. The format of the M_PROTO message block is as follows:

```
struct T_conn_ind {
        long    PRIM_type;    /* always T_CONN_IND */
        long    SRC_length;   /* source addr length - see note in sec. 1.4*/
        long    SRC_offset;   /* source addr offset */
        long    OPT_length;   /* options length - see note in sec. 1.4 */
        long    OPT_offset;   /* options offset */
        long    SEQ_number;   /* sequence number - see note in sec. 1.4 */
}
```

Where *PRIM_type* identifies the primitive type. *SRC_length* is the length of the source address and *SRC_offset* is the offset from the beginning of the M_PROTO message block where the source address begins. Similarly, *OPT_length* and *OPT_offset* describe the location of the requested options associated with the primitive. *SEQ_number* should be an unique number other than -1 to identify the connect indication. The proper alignment of the source address and options in the M_PROTO message block is not guaranteed.

2.2.2.2 T_CONN_CON - connection confirm.

This primitive indicates to the user that a connect request has been confirmed on the specified responding address. The format of this message is one M_PROTO message block followed by zero or more M_DATA blocks if any user data is associated with the primitive. The format of the M_PROTO message block is as follows:

```
struct T_conn_con {
        long    PRIM_type;    /* always T_CONN_CON */
        long    RES_length;   /* responding addr length - see note in sec. 1.4*/
        long    RES_offest;   /* responding addr offset */
        long    OPT_length;   /* options length - see note in sec. 1.4 */
        long    OPT_offset;   /* options offset */
}
```

Where *PRIM_type* identifies the primitive type. *RES_length* is the length of the responding address that the connection was accepted on and *RES_offset* is the offset from the beginning of the M_PROTO message block where the responding address begins. Similarly, *OPT_length* and *OPT_offset* describe the size and location of the confirmed options associated with the primitive. The proper alignment of the responding address and options in the M_PROTO message block is not guaranteed.

2.2.2.3 T_DISCON_IND - disconnect indication.

This primitive indicates to the user that either a request for connection has been denied or an existing connection has been disconnected. The format of this message is one

M_PROTO message block possibly followed by one or more M_DATA message blocks if there is any user data associated with the primitive. The format of the M_PROTO message block is as follows:

```
struct T_discon_ind {
        long PRIM_type;              /* always T_DISCON_IND */
        long DISCON_reason;     /* disconnect reason - see note in sec. 1.4 */
        long SEQ_number;          /* sequence number - see note in sec. 1.4 */
}
```

Where *PRIM_type* identifies the primitive type and *DISCON_reason* is the reason for disconnect. The reason codes are protocol specific. *SEQ_number* is the sequence number which identifies which connect indication was denied, or it is -1 if the provider is disconnecting an existing connection. The *SEQ_number* is only meaningful when this primitive is sent to a passive user who has the corresponding connect indication outstanding. It allows the transport user to identify which of its outstanding connect indications is associated with the disconnect.

2.2.2.4 T_DATA_IND - data indication.

This primitive indicates to the transport user that this message contains a transport interface data unit. One or more transport interface data units form a transport service data unit. This primitive has a mechanism which indicates the beginning and end of a transport service data unit. However, not all transport providers support the concept of a transport service data unit, as noted in section 2.1.2.1. The format of the message is one M_PROTO message block followed by zero or more M_DATA message blocks where each M_DATA message block, except for the last, must contain at least one byte of data. The format of the M_PROTO message block is as follows:

```
struct T_data_ind {
        long PRIM_type;          /* always T_DATA_IND */
        long MORE_flag;         /* indicates more data in TSDU */
}
```

Where *PRIM_type* identifies the primitive type. *MORE_flag*, when greater than zero, indicates that the next T_DATA_IND primitive is also part of this transport service data unit.

If a TSDU spans multiple T_DATA_IND message blocks, then an ETSDU may be placed in between two T_DATA_IND message blocks. Once an ESTDU is started, then the ETSDU must be completed before any T_DATA_IND message blocks defining a TSDU is resumed.

2.2.2.5 T_EXDATA_IND - expedited data indication.

This primitive indicates to the transport user that this message contains an expedited transport interface data unit. One or more expedited transport interface data units form an expedited transport service data unit. This primitive has a mechanism which indicates the beginning and end of an expedited transport service data unit. However, not all transport providers support the concept of an expedited transport service data unit, as noted in section 2.1.2.1. The format of the message is one M_PROTO message block followed by one or more M_DATA message blocks containing at least one byte of data. The format of the M_PROTO message block is as follows:

```
struct T_exdata_ind {
        long PRIM_type;          /* always T_EXDATA_IND */
        long MORE_flag;          /* indicates more data in ETSDU */
}
```

Where *PRIM_type* identifies the primitive type. *MORE_flag*, when greater than zero, indicates that the next T_EXDATA_IND primitive is also part of this expedited transport service data unit.

2.2.2.6 T_ORDREL_IND - orderly release indication.

This primitive indicates to the transport user that the user on the other side of the connection is finished sending data. This primitive is only supported by the transport provider if it is of type T_COTS_ORD. The format of the message is one M_PROTO message block. The format of the M_PROTO message block is as follows:

```
struct T_ordrel_ind {
        long PRIM_type;          /* always T_ORDREL_IND */
}
```

Where *PRIM_type* identifies the primitive type.

2.3 Connectionless-Mode Transport Primitives

The following transport primitives pertain only to the connectionless-mode transport service.

2.3.1 User-Originated Primitives

2.3.1.1 T_UNITDATA_REQ - unitdata request.

This primitive requests that the transport provider send the specified datagram to the specified destination. The format of the message is one M_PROTO message block followed by zero or more M_DATA message blocks where each M_DATA message block contains zero or more bytes of data. The format of the M_PROTO message block is as follows:

```
struct T_unitdata_req {
        long    PRIM_type;    /* always T_UNITDATA_REQ */
        long    DEST_length;  /* dest addr length */
        long    DEST_offset;  /* dest addr offset */
        long    OPT_length;   /* options length */
        long    OPT_offset;   /* options offset */
}
```

Where *PRIM_type* identifies the primitive type. *DEST_length* is the length of the destination address and *DEST_offset* is the offset from the beginning of the M_PROTO message block where the destination address begins. Similarly, *OPT_length* and *OPT_offset* describe the location of the requested options associated with the primitive. The proper alignment of the destination address and options in the M_PROTO message block is not guaranteed. The destination address and options in the M_PROTO message block are, however, aligned the same as they were received from the transport user.

This primitive does not require any acknowledgment. If an non-fatal error occurs, it is the responsibility of the transport provider to report it via the T_UDERROR_IND indication. Fatal errors are indicated via a 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 indicates one of the following unrecoverable protocol conditions:

- — The transport service interface was found to be in an incorrect state.
- — The amount of transport user data associated with the primitive defines an transport service data unit larger than that allowed by the transport provider.

2.3.2 Provider-Originated Primitives

2.3.2.1 T_UNITDATA_IND - unitdata indication.

This primitive indicates to the transport user that a datagram has been received from the specified source address. The format of the message is one M_PROTO message block followed by zero or more M_DATA message blocks where each M_DATA message block contains at least one byte of data. The format of the M_PROTO message block is as follows:

```
struct T_unitdata_ind {
        long    PRIM_type;     /* always T_UNITDATA_IND */
        long    SRC_length;    /* source addr length  - see note in sec. 1.4 */
        long    SRC_offset;    /* source addr offset */
        long    OPT_length;    /* options length - see note in sec. 1.4  */
        long    OPT_offset;    /* options offset */
}
```

Where *PRIM_type* identifies the primitive type. *SRC_length* is the length of the source address and *SRC_offset* is the offset from the beginning of the M_PROTO message block where the source address begins. Similarly, *OPT_length* and *OPT_offset* describe the location of the requested options associated with the primitive. The proper alignment of the source address and options in the M_PROTO message block is not guaranteed.

2.3.2.2 T_UDERROR_IND - unitdata error indication.

This primitive indicates to the transport user that a datagram with the specified destination address and options produced an error. The format of this message is one M_PROTO message block. The format of the M_PROTO message block is as follows:

```
struct T_uderror_ind {
        long    PRIM_type;     /* always T_UDERROR_IND */
        long    DEST_length;   /* destination addr length - see note in sec. 1.4 */
        long    DEST_offset;   /* destination addr offset */
        long    OPT_length;    /* options length - see note in sec. 1.4  */
        long    OPT_offset;    /* options offset */
        long    ERROR_type;    /* error type */
}
```

Where *PRIM_type* identifies the primitive type. *DEST_length* is the length of the destination address and *DEST_offset* is the offset from the beginning of the M_PROTO message block where the destination address begins. Similarly, *OPT_length* and *OPT_offset* describe the location of the requested options associated with the primitive. *ERROR_type* defines the protocol dependent error code. The proper alignment of the destination address and options in the M_PROTO message block is not guaranteed.

2.4 Note about Structure Elements

Although the structure elements in the Transport Provider Interface are declared as **long** data types, the value the transport provider assigns to those elements that refer to this note must not be greater than the maximum value of an **int** data type because the corresponding user level structure element is declared as an **int**.

2.5 Overview of Error Handling Capabilities

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

2.5.1 Non-fatal Errors

The non-fatal errors are those that a transport user can correct, and are reported in the form of an error acknowledgment to the appropriate primitive in error. Only those primitives which require acknowledgments may generate a non-fatal error acknowledgment. These acknowledgments always report a syntactical error in the specified primitive when the transport provider receives the primitive. The primitive descriptions above define those primitives and rules regarding the acknowledgment of them. These errors are reported to the transport user via the T_ERROR_ACK primitive, and give the transport user the option of reissuing the transport service primitive that caused the error. The T_ERROR_ACK primitive also indicates to the transport user that no action was taken by the transport provider upon receipt of the primitive which caused the error.

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

The allowable errors that can be reported on the receipt of a transport initiated primitive are presented in the description of the appropriate primitives.

2.5.2 Fatal Errors

Fatal errors are those which can not be corrected by the transport user, or those errors which result in an uncorrectable error in the interface or in the transport provider.

The most common of these errors are listed under the appropriate primitives. The transport provider should issue fatal errors only if the transport user can not correct the condition which caused the error or if the transport provider has no means of reporting a transport user correctable error. If the transport provider detects an uncorrectable non-protocol error internal to the transport provider, the provider should issue a fatal error to the user.

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

2.6 Transport Service Interface Sequence of Primitives

The allowable sequence of primitives are described in the state diagrams and tables in section 4 for both the connection-mode and connectionless-mode transport services. The following are rules regarding the maintenance of the state of the interface:

- It is the responsibility of the transport provider to keep record of the state of the interface as viewed by the transport user.
- The transport 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 T_BIND_REQ primitive) before the transport provider may view it as an active *stream*.
- If the transport provider sends a M_ERROR upstream, it should also drop any further messages received on its write side of the *stream*.

The following rules apply only to the connection-mode transport services.

- A transport connection release procedure can be initiated at any time during the transport connection establishment or data transfer phase.
- The state tables for the connection-mode transport service providers include the management of the sequence numbering when a transport provider sends multiple T_CONN_IND requests without waiting for the response of the previously sent indication. It is the responsibility of the transport providers not to change state until all the indications have been responded to, therefore the provider should remain in the T_WRES_CIND state while there are any outstanding connect indications pending response. The provider should change state appropriately when all the connect indications have been responded to.
- The only time the state of a transport service interface of a *stream* may be transferred to another *stream* is when it is indicated in a T_CONN_RES primitive. The following rules then apply to the cooperating *streams*:
  - — The *stream* which is to accept the current state of the interface must be bound to an appropriate protocol address and must be in the idle state.
  - — The user transferring the current state of a *stream* must have correct permissions for the use of the protocol address bound to the accepting *stream*.
  - — The *stream* which transfers the state of the transport interface must be placed into an appropriate state after the completion of the transfer.

2.7 Precedence of Transport Interface Primitives on a *Stream*

The following rules apply to the precedence of transport interface primitives with respect to their position on a *stream*[13]:

- The transport provider has responsibility for determining precedence on its *stream* write queue, as per the rules in section 5. The appendix specifies the rules for precedence for both the connection-mode and connectionless-mode transport services.
- The transport user has responsibility for determining precedence on its *stream* read queue, as per the rules in section 5.
- All primitives on the *stream* are assumed to be placed on the queue in the correct sequence as defined above.

The following rules apply only to the connection-mode transport services.

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

---

13. The *stream* queue which contains the transport user initiated primitives is referred to as the *stream* write queue. The *stream* queue which contains the transport provider initiated primitives is referred to as the *stream* read queue.

2.8 Rules for Flushing Queues

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

- The transport providers must be aware that they will receive M_FLUSH messages 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 transport provider must send up a M_FLUSH message to flush both the read and write queues after receiving a successful T_UNBIND_REQ message and prior to issuing the T_OK_ACK primitive.

The following rules pertain only to the connection-mode transport providers.

- If the interface is in the T_DATA_XFER, T_WIND_ORDREL or T_WACK_ORDREL state, the transport provider must send up a M_FLUSH message to flush both the read and write queues before sending up a T_DISCON_IND.
- If the interface is in the T_DATA_XFER, T_WIND_ORDREL or T_WACK_ORDREL state, the transport provider must send up a M_FLUSH message to flush both the read and write queues after receiving a successful T_DISCON_REQ message and before issuing the T_OK_ACK primitive.

3. Mapping of Transport Primitives to OSI

The following table maps those transport primitives as seen by the transport provider to the STREAMS message types used to realize the primitives and to the ISO IS 8072 and IS 8072/DAD1 transport service definition primitives.

Transport Primitives	Stream Message Types	IS 8072 Transport Primitives
T_CONN_REQ	M_PROTO	T-CONNECT request
T_CONN_IND	M_PROTO	T-CONNECT indication
T_CONN_RES	M_PROTO	T-CONNECT response
T_CONN_CON	M_PROTO	T-CONNECT confirm
T_DATA_REQ	M_PROTO	T-DATA request
T_DATA_IND	M_PROTO	T-DATA indication
T_EXDATA_REQ	M_PROTO	T-EXPEDITED-DATA request
T_EXDATA_IND	M_PROTO	T-EXPEDITED-DATA indication
T_DISCON_REQ	M_PROTO	T-DISCONNECT request
T_DISCON_IND	M_PROTO	T-DISCONNECT indication
T_UNITDATA_REQ	M_PROTO	T-UNITDATA request
T_UNITDATA_IND	M_PROTO	T-UNITDATA indication
T_ORDREL_REQ	M_PROTO	not defined in ISO
T_ORDREL_IND	M_PROTO	not defined in ISO
T_BIND_REQ	M_PROTO	not defined in ISO
T_BIND_ACK	M_PCPROTO	not defined in ISO
T_UNBIND_REQ	M_PROTO	not defined in ISO
T_OK_ACK	M_PCPROTO	not defined in ISO
T_ERROR_ACK	M_PCPROTO	not defined in ISO
T_INFO_REQ	M_PCPROTO	not defined in ISO
T_INFO_ACK	M_PCPROTO	not defined in ISO
T_UDERR_IND	M_PROTO	not defined in ISO
T_OPTMGMT_REQ	M_PROTO	not defined in ISO
T_OPTMGMT_ACK	M_PCPROTO	not defined in ISO
T_ADDR_REQ	M_PROTO	not defined in ISO
T_ADDR_ACK	M_PCPROTO	not defined in ISO

Figure 2. Mapping ISO IS 8072 and IS 8072/DAD1 to Kernel-level Transport Service Primitives

4. Allowable Sequence of Transport Service Primitives

The following tables describe the possible events that may occur on the interface and the possible states as viewed by the transport user that the interface may enter due to an event. The events map directly to the transport service interface primitives as described in section 2.

Possible States

state	abbreviation	description	service type
sta_0	unbnd	unbound	T_COTS, T_COTS_ORD, T_CLTS
sta_1	w_ack b_req	awaiting acknowledgment of T_BIND_REQ	T_COTS, T_COTS_ORD, T_CLTS
sta_2	w_ack u_req	awaiting acknowledgment of T_UNBIND_REQ	T_COTS, T_COTS_ORD, T_CLTS
sta_3	idle	idle - no connection	T_COTS, T_COTS_ORD, T_CLTS
sta_4	w_ack op_req	awaiting acknowledgment of T_OPTMGMT_REQ	T_COTS, T_COTS_ORD, T_CLTS
sta_5	w_ack c_req	awaiting acknowledgment of T_CONN_REQ	T_COTS, T_COTS_ORD
sta_6	w_con c_req	awaiting confirmation of T_CONN_REQ	T_COTS, T_COTS_ORD
sta_7	w_res c_ind	awaiting response of T_CONN_IND	T_COTS, T_COTS_ORD
sta_8	w_ack c_res	awaiting acknowledgment of T_CONN_RES	T_COTS, T_COTS_ORD
sta_9	data_t	data transfer	T_COTS, T_COTS_ORD
sta_10	w_ind or_rel	awaiting T_ORDREL_IND	T_COTS_ORD
sta_11	w_req or_rel	awaiting T_ORDREL_REQ	T_COTS_ORD
sta_12	w_ack dreq6	awaiting acknowledgment of T_DISCON_REQ	T_COTS, T_COTS_ORD
sta_13	w_ack dreq7	awaiting acknowledgment of T_DISCON_REQ	T_COTS, T_COTS_ORD
sta_14	w_ack dreq9	awaiting acknowledgment of T_DISCON_REQ	T_COTS, T_COTS_ORD
sta_15	w_ack dreq10	awaiting acknowledgment of T_DISCON_REQ	T_COTS, T_COTS_ORD
sta_16	w_ack dreq11	awaiting acknowledgment of T_DISCON_REQ	T_COTS, T_COTS_ORD

Figure 3. Kernel Level Transport Interface States

Variables and Outputs

The following describes the variables and outputs used in the state tables.

variable	description
q	queue pair pointer of current *stream*
rq	queue pair pointer of responding *stream* as described in the T_CONN_RES primitive
outcnt	counter for the number of outstanding connection indications not responded to by the transport user

Figure 4. State table Variables

output	description
[1]	outcnt = 0
[2]	outcnt = outcnt + 1
[3]	outcnt = outcnt - 1
[4]	pass connection to queue as indicated in the T_CONN_RES primitive

Figure 5. State Table Outputs

Outgoing Events

The following outgoing events are those which are initiated from the transport service user. They either make requests of the transport provider or respond to an event of the transport provider.

event	description	service type
bind_req	bind request	T_COTS, T_COTS_ORD, T_CLTS
unbind_req	unbind request	T_COTS, T_COTS_ORD, T_CLTS
optmgmt_req	options mgmt request	T_COTS, T_COTS_ORD, T_CLTS
conn_req	connection request	T_COTS, T_COTS_ORD
conn_res	connection response	T_COTS, T_COTS_ORD
discon_req	disconnect request	T_COTS, T_COTS_ORD
data_req	data request	T_COTS, T_COTS_ORD
exdata_req	expedited data request	T_COTS, T_COTS_ORD
ordrel_req	orderly release request	T_COTS_ORD
unitdata_req	unitdata request	T_CLTS

Figure 6. Kernel Level Transport Interface Outgoing Events

Incoming Events

The following incoming events are those which are initiated from the transport provider. They are either confirmations of a request or are indications to the transport user that an event has occurred.

event	description	service type
bind_ack	bind acknowledgment	T_COTS, T_COTS_ORD, T_CLTS
optmgmt_ack	options mgmt acknowledgment	T_COTS, T_COTS_ORD, T_CLTS
error_ack	error acknowledgment	T_COTS, T_COTS_ORD, T_CLTS
ok_ack1	ok acknowledgment outcnt == 0	T_COTS, T_COTS_ORD, T_CLTS
ok_ack2	ok acknowledgment outcnt == 1, q == rq	T_COTS, T_COTS_ORD,
ok_ack3	ok acknowledgment outcnt == 1, q != rq	T_COTS, T_COTS_ORD,
ok_ack4	ok acknowledgment outcnt > 1	T_COTS, T_COTS_ORD,
conn_ind	connection indication	T_COTS, T_COTS_ORD
conn_con	connection confirmation	T_COTS, T_COTS_ORD
data_ind	data indication	T_COTS, T_COTS_ORD
exdata_ind	expedited data indication	T_COTS, T_COTS_ORD
ordrel_ind	orderly release indication	T_COTS_ORD
discon_ind1	disconnect indication outcnt == 0	T_COTS, T_COTS_ORD
discon_ind2	disconnect indication outcnt == 1	T_COTS, T_COTS_ORD
discon_ind3	disconnect indication outcnt > 1	T_COTS, T_COTS_ORD
pass_conn	pass connection	T_COTS, T_COTS_ORD
unitdata_ind	unitdata indication	T_CLTS
uderror_ind	unitdata error indication	T_CLTS

Figure 7. Kernel Level Transport Interface Incoming Events

Transport Service State Tables

The following tables describes the possible next states the interface may enter given a current state and event.

The contents of each box represent the next state given the current state (column) and the current incoming or outgoing event (row). An empty box represents a state/event combination that is invalid. Along with the next state, each box may include an action. The transport provider must take the specific actions in the order specified in the state table.

event \ state	sta_0 unbnd	sta_1 w_ack b_req	sta_2 w_ack u_req	sta_3 idle	sta_4 w_ack op_req
bind_req	sta_1				
unbind_req				sta_2	
optmgmt_req				sta_4	
bind_ack		sta_3 [1]			
optmgmt_ack					sta_3
error_ack		sta_0	sta_3		sta_3
ok_ack1			sta_0		

Figure 8. Initialization State Table

event \ state	sta_0 unbnd	sta_3 idle	sta_5 w_ack c_req	sta_6 w_con c_req	sta_7 w_res c_ind	sta_8 w_ack c_res	sta_9 data_t	sta_10 w_ind or_rel **	sta_11 w_req or_rel **	sta_12 w_ack dreq6	sta_13 w_ack dreq7	sta_14 w_ack dreq9	sta_15 w_ack dreq10	sta_16 w_ack dreq11
conn_req		sta_5												
conn_res					sta_8									
discon_req				sta_12	sta_13		sta_14	sta_15	sta_16					
data_req							sta_9		sta_11					
exdata_req							sta_9		sta_11					
** ordrel_req							sta_10		sta_3					
conn_ind		sta_7 [2]			sta_7 [2]									
conn_con				sta_9										
data_ind							sta_9	sta_10						
exdata_ind							sta_9	sta_10						
** ordrel_ind							sta_11	sta_3						
discon_ind1				sta_3			sta_3	sta_3	sta_3					
discon_ind2					sta_3 [3]									
discon_ind3					sta_7 [3]									
error_ack			sta_3			sta_7				sta_6	sta_7	sta_9	sta_10	sta_11
ok_ack1			sta_6							sta_3		sta_3	sta_3	sta_3
ok_ack2						sta_9 [3]					sta_3 [3]			
ok_ack3						sta_3 [3][4]					sta_3 [3]			
ok_ack4						sta_7 [3][4]					sta_7 [3]			
pass_conn	sta_9	sta_9												

** Only supported if service is type T_COTS_ORD

Figure 9. Connection/Release/Data-Transfer State Table for Connection Oriented Service

state / event	sta_3 idle
unitdata_req	sta_3
unitdata_ind	sta_3
uderror_ind	sta_3

Figure 10. Data-Transfer State Table for Connectionless Service

5. Transport Primitive Precedence

The following describes the precedence of the transport primitives for both the *stream*[14] write and read queues.

Y \ X	t_addr_req	t_conn_req	t_conn_res	t_discon_req	t_data_req	t_exdata_req	t_bind_req	t_unbind_req	t_info_req	t_unitdata_req	t_optmgmt_req	t_ordrel_req
t_addr_req												
t_conn_req				4								
t_conn_res				3								
t_discon_req												
t_data_req				5	1	2						1
t_exdata_req				5	1	1						1
t_bind_req												
t_unbind_req												
t_info_req												
t_unitdata_req										1		
t_optmgmt_req												
t_ordrel_req				5								

Key

blank: not applicable / queue should be empty

1 : X has no precedence over Y

2 : X has precedence over Y

3 : X has precedence over Y and Y must be removed

4 : X has precedence over Y and both X and Y must be removed

5 : X may have precedence over Y (choice of user) and if X does, then it is the same as 3

Figure 11. *Stream* Write Queue Precedence Table

14. The *stream* queue which contains the transport user initiated primitives is referred to as the *stream* write queue. The *stream* queue which contains the transport provider initiated primitives is referred to as the *stream* read queue.

Y \ X	t_addr_ack	t_conn_ind	t_conn_con	t_discon_ind	t_data_ind	t_exdata_ind	t_info_ack	t_bind_ack	t_error_ack	t_ok_ack	t_unitdata_ind	t_uderror_ind	t_optmgmt_ack	t_ordrel_ack
t_addr_ack														
t_conn_ind				4										
t_conn_con				3	1	1								
t_discon_ind		1							2	2				
t_data_ind				5	1	2				1				1
t_exdata_ind				5	1	1				1				1
t_info_ack														
t_bind_ack		1												
t_error_ack		1	1	1	1	1								
t_ok_ack		1	1	1	1	1								
t_unitdata_ind									2		1	2	2	
t_uderror_ind									1		1	1	1	
t_optmgmt_ack		1									1	1		
t_ordrel_ack		1		5					2	2				

Key

blank: not applicable / queue should be empty

1 : X has no precedence over Y

2 : X has precedence over Y

3 : X has precedence over Y
and Y must be removed

4 : X has precedence over Y and
both X and Y must be removed

5 : X may have precedence over Y
(choice of user) and if X does, then
it is the same as 3

Figure 12. *Stream* Read Queue Precedence Table

PLEASE DISCARD THIS PAGE!!!

Table of Contents

List of Figures