HP-JetSend Transport Independent API ERS

Version 3.21

Allen Baker

Hewlett-Packard Company
INSO R&D Lab

November 7, 1996

1. Revisions

1.1 Version 3.21 (7nov96)

- Changed TCP and UDP port from 9120 to IANA assigned 1782

1.2 Version 3.20 (25oct96)

- Changed semantics of ETshutdownGraceful shutdown option.
- Deleted all ETrplyIdNetwork* reply Ids.
- xptInitialize() now returns a TcCode instead of void.

1.3 Version 3.10 (27aug96)

- Changed parameter to xptGetNextEvent() from a pointer to a Trply to a pointer to a pointer to a Trply. This eliminates a data copy on all received packets.

1.4 Version 3.01 (15aug96)

- Specified exact use of Taddr parameters.
- Specified (and sometimes changed so gateways will work) how port fields of Taddr parameters are interpreted.
- Added section on well-known port numbers. Includes constant definitions and description of how they are used with multi-port devices like JetDirect EX boxes.
- Added error code ETcCodeInvalidLocalPort.
- Moved responsibility for finding right local address of multi-homed device back into TIAPI.

1.5 Version 3.00 (30jul96)

- Removed xptEnumerate()
- Added typedefs TrplyNwkStat and TnwkStat
- Required OSP to specify the local address through which TIAPI should make a connection:
  - ⇒ Added prmLocalAddrPtr to xptCall() parameter list.
  - ⇒ Added mbrLocalAddr to TrplyActiveConnect.
  - ⇒ Added mbrLocalAddr to TrplyPassiveConnect.
  - ⇒ Added prmLocalAddrPtr to xptSendDatagram() parameter list.
  - ⇒ Added mbrLocalAddr to TrplySendDatagram.
  - ⇒ Added mbrLocalAddr to TrplyReceiveDatagram.
  - ⇒ Added comments that specify how OSP should handle multi-homed local hosts.
- Added comments to specify how the Taddr mbrPort field (or mbrSocket field in the case of IPX address) is used by TIAPI in functions that take a Taddr parameter.
- Changed comments in some Semantics sections.
- Added draft comments about how TIAPI handles multi-port EX box addressing.

1.6 Version 2.02 (26jul96)

- Added name section to each function and reply description.

1.7 Version 2.01 (17jul96)

- Fixed compile error in xptXlateTIAddr()
- Cleared up confusion about state of Tbcb* parameters in xptSend(), xptReceive(), xptSendDatagram(), and xptReceiveDatagram().

1.8 Version 2.00 (16jul96)

- Added function xptXlateTIAddr() and coresponding reply.

1.9 Version 1.80 (10jul96)

- Added function xptNetworkEvent() to API.

1.10 Version 1.73 (28jun96)

- Added ETcCodeInvalidConn as a possible return code for all functions that take a connection handle as a parameter.
- Added ETcCodeConnClosed and ETcCodeConnFailed as possible cCodes to TrplyDisconnect.
- Changed cnsUnixPath from 18 to 40

1.11 Version 1.72 (14jun96)

- removed in and out parameter class specifiers.

1.12 Version 1.71 (11jun96)

- Added #include "tops_buf.h"
- removed conditional compile of domain enumerations.
- removed domain enumeration equivalences to sockets AF_* constants.

1.13 Version 1.70 (10jun96)

- Changed xptEnumerate into an asynchronous call since it has to be implemented asynchronously on at least one platform - Win32.
- Changed MS OS back slashes to Unix forward slashes in the include statements that get compiled on a Unix platform.
- Added send and receive datagram functionality as required for gateway broadcasts.
- changed TrqstHdl from void* to Ts32.
- pulled union field out of Taddr typedef and put into its own typedef named Taddress. Added a member field of type Taddress back into Taddr typedef.
- Made it an error to make multiple listen requests on the same local address.

1.14 Version 1.62 (4jun96)

- Removed max message size from enumeration of networks and made it a return field in the connect replies.

- Set domain enums equal to sockets.h AF_ values. Made them conditionally compiled.

1.15 Version 1.61 (24may96)

- Changed void* data parameter in send and receive functions and their corresponding replies to Tbcd*.

1.16 Version

- Added Introduction.
- Minor spelling and wording changes.

1.17 Version 1.60 (16may96)

- Fixed misspelling of ETcCodeInvalidConn.
- Added "up:" for UnixPath and "lb:" for LoopBack to description of networks in xptXlateOSPAddr().
- Added xptShutdown().
- Added xptCancelRequest().
- Removed xptCancelListen() and TrplyCancelListen.
- Added ETcCodeRequestCancelled as a possible value to all reply TcCode members.
- Replaced references to R-UDP, R-IPX, etc. with references to RMTP.

1.18 Version 1.51 (9may96)

- Removed xptXlateTIAddr() and corresponding reply TrplyXlateTIAddr which provided the unneeded service to OSP of translating a TIAPI address to an OSP address.
- Removed parameter prmOSPAddrPtr from xptXlateOSPAddr().
- Changed TrplyXlateOSPAddr member mbrTIAddrPtr to mbrTIAddr. mbrTIAddr is a structure containing the address and mbrTIAddrPtr was a pointer to a structure.

1.19 Version 1.50 (8may96)

- Removed all references to MLC.
- Removed redundant type definitions for Tchar.
- Removed unused typedef for Tbool.
- Format changes.

1.20 Version 1.40

- Changed data sizes in Send and Receive request/reply to signed to map directly to Sockets and TLI.
- Added function RunTIAPI() which passes to TIAPI a thread for checking for network events and performing other internal housekeeping.
- Added xptInitialize() to initialize TIAPI.
- Removed superfluous struct construct from Trply typedef.
- Made separate replies for connects that result from call to xptConnect() and xptListen().
  - ⇒ Removed TrplyConnect
  - ⇒ Added TrplyActiveConnect and TrplyPassiveConnect which correspond to xptConnect() and xptListen() respectively.

1.21 Version 1.31

- Added Unix domain and address for testing.
- Renamed Tprotocol to TsrvcType.
- Made header file conditionally compilable.
- Replaced literal array sizes with constants.

1.22 Version 1.30

- Replaced datagram calls and replies with a protocol parameter in xptCall() and xptListen(). Protocol parameter specifies STREAM or MESSAGE protocol.
- Added more platforms.
- Removed compiler errors from API and made Unix C compiler compatible.
- Added loopback domain for testing.

1.23 Version 1.20

- Added datagram service Open and Close functions.
- Added Semantics sections to function headers.
- Added additional error and status codes.
- Changed function return types from Boolean to Status.
- Eliminated Status type out parameters from functions.

1.24 Version 1.10

- Changed function and event prefix string from "TI" to "xpt" (meaning transport or xport) to match coding style that other engineers are using in other modules.
- Added request handle parameter to all functions that have a corresponding asynchronous reply.
- Included matching request handle in replies.
- Replaced the word "event" with the word "reply" in those cases where "reply" more accurately describes what is going on.
- Changed storage for IPv6 addresses from 64 to 128 bits.

1.25 Version 1.00

- Created this document by moving this information out of the **HP-JetSend Network Architecture** document.

2. Patents

The protocol described in this document is protected under the following international patents which are owned by the Hewlett-Packard Company:

1. DE69802792D1
2. DE69802792T2
3. DE69835314D1
4. DE69835314T2
5. EP0872991A2
6. EP0872991A3
7. EP0872991B1
8. EP0886411A2
9. EP0886411A3
10. EP0976058A1
11. EP0976058B1
12. US6202096 B1
13. US6502000
14. US6721286
15. WO1998047076A1

3. Introduction

An External Reference Specification (ERS) specifies its target object's look, feel, and functionality from an external perspective. The external perspective is a view of the target object from the outside and does not include requirements specifications, the target object's internal architectural and design details, or the target object's architectural relationships with other subsystems and modules (the target object's relationships to other subsystems and modules is described in the Architecture Specification). The external perspective includes all of the target object's hardware and software interfaces, protocols, performance characteristics, functionality, and operational procedures. Interfaces are inputs and outputs like those to and from users and other systems, subsystems and modules. An ERS describes its target object's interfaces and functionalities as a whole, without describing how those interfaces and functionalities are allocated to subsystems and modules within the target object.

The target object for this ERS is the Transport Independent API (TIAPI).

4. Application Program Interface (API)

```
/*
$Header: appl_tiapi.h,v 1.8 96/10/30 17:21:14 kdahl Exp $
*/

#if !defined(_IGNORE_TIAPI_H)
   #define _IGNORE_TIAPI_H

   #ifdef __cplusplus
      extern "C" {
   #endif


   /* ================================================================ */
   /* ||||||||| (c) Copyright Hewlett-Packard Company 1996. |||||||||| */
   /* |||||||||             All Rights Reserved.            |||||||||| */
   /* ================================================================ */


   /* ================================================================ */
   /* ||||||||||||||||||||||| Description ||||||||||||||||||||||||||| */
   /* ================================================================ */


   /*
   ================================================

   - TAB size is 3.

   - Version information is maintained in TIAPIERS.Doc

   The TransportIndependentAPI (TIAPI) provides the HP-JetSendSessionProtocol
   (OSP) with a common API to the transport service provider(s) within the
   device.  TIAPI is a glue layer that attaches the OSP to the
   device-specific, network-connectivity implementation.

   The TIAPI is a service, not a protocol.  It simply provides a single
   common API to multiple transports.  The protocols used under the TIAPI
   are the ones defined by the transports that are installed and active
   on the host platform.  The transport is selected by the OSP when it
   makes a TIAPI function call.

   HP-JetSend could have chosen an industry standard transport API such as
   Berkeley Sockets or TLI as the TIAPI.  But neither of those fits into
   the HP-JetSend core architecture.  The HP-JetSend architecture requires that
all
   TIAPI calls be asynchronous.  Also, the architecture defines a specific
   method for passing information from TIAPI to OSP that neither Berkeley
   Sockets or TLI provides.  TIAPI is simpler than Berkeley Sockets and
   TLI.  This allows The HP-JetSend Protocol Stack (TOPS) team to write OSP
code
   that is free of Transport API complexities that may obscure the OSP's
   own algorithms.

   The HP-JetSend session protocol resides in the OSP.  That is the entity that
   manages HP-JetSend session protocol issues like routing to TOPS modules
   based on OSP port number.  It is the user of the TIAPI.
```

```
This file defines the TIAPI.  The API consists of all the constant,
type, function and event (events are called replies in this API)
definitions that the OSP needs to access the transport service
providers within the local host.

The semantics of processing each request are presented with each
function.  For clarity and simplicity, the semantics are expressed in
terms of Berkeley Sockets.  The semantics intentionally don't obscure
the central algorithm with error and special case handling.  Semantics
are provided to further illuminate the functionality provided, not to
be an exhaustive detailed internal design or example implementation of
the functions.  I named the section "Semantics:" instead of
"Algorithm:" to highlight this fact.
---------------------------------------------- */



/* ================================================================ */
/* |||||||||||||||||||| Included Files |||||||||||||||||||||||||| */
/* ================================================================ */

#ifdef __cplusplus
   }
#endif

#include    "appl_buf.h"


#if     (defined(PLATFORM_WIN95) || defined(PLATFORM_WINNT))
   #include <sys\types.h>
   #include <winsock.h>

#elif   defined(PLATFORM_Unix32)
   #include <sys/types.h>
   #include <sys/socket.h>

#endif

#ifdef __cplusplus
   extern "C" {
#endif

/* ================================================================ */
/* |||||||||||||||||||| Symbols and Constants |||||||||||||||||||| */
/* ================================================================ */


/*
================================================
Placeholder definitions for unknown things
---------------------------------------------- */
#define  TBD_I          1  /* unknown integer value */
/*
================================================
Array size constants
---------------------------------------------- */
#define  cnsUnixPath    40
#define  cnsINETIPAddr  16
#define  cnsINETPort    2
#define  cnsSIPXNet     4
#define  cnsSIPXNode    6
#define  cnsSIPXSock    2
```

```
/*
================================================
Well known transport port numbers.  TIAPI uses three well-known ports
for each network domain.  One is the datagram transport port over which
RMTP will run (in the INET domain this is a UDP port).  The second is
the reliable-connection-oriented transport port (in the INET domain
this is a TCP port).  The third is the datagram port that OSP gateway
messages are sent and received through (in the INET domain this is a
UDP port).

For each domain, the well-known ports are registered with the
appropriate authority.  In the case of the INET domain, the ports are
registered through the Internet Assigned Numbers Authority (IANA).
For the SIPX domain, the numbers are registered through Novell.

When TIAPI announces the availability of a local network address to OSP
(via the TrplyNwkStat reply), it fills in the port field of the network
address with the datagram transport port over which RMTP will run.
This is the HP-JetSend contact address that should be displayed to the user
of the local host as the address of his/her HP-JetSend device.

For multi-port boxes like JetDirect EX to provide pass-through support
for HP-JetSend, a unique set of the three well-known transport ports is
needed for each of the parallel ports on the box.  TIAPI is responsible
for finding out which parallel port its host is attached to (if any)
and using the correct set of well-known transport ports for it.  See
the HP-JetSend Network ARS for more information about "Multi-Port Network-
Parallel Gateway Addressing".

This section defines the well-known port numbers.  For each combination
of domain and multi-port EX box port, there are 3 well-known port
numbers defined.  One of the sets of three numbers in each domain is
considered the default set.  That set is used when there is no multi-
port box in use, such as when the JetDirect MIO card is in use.
------------------------------------------------ */

                  /* INET Domain */

#define  cnsRmtpUdp0    1782     /* EX box port 1, Non-EX box, default */
#define  cnsTcp0        1782     /* EX box port 1, Non-EX box, default */
#define  cnsUdp0        9130     /* EX box port 1, Non-EX box, default */

#define  cnsRmtpUdp1    9121     /* EX box port 2 */
#define  cnsTcp1        9121     /* EX box port 2 */
#define  cnsUdp1        9131     /* EX box port 2 */

#define  cnsRmtpUdp2    9122     /* EX box port 3 */
#define  cnsTcp2        9122     /* EX box port 3 */
#define  cnsUdp2        9132     /* EX box port 3 */

#define  cnsRmtpUdp3    9123     /* EX box port 4 */
#define  cnsTcp3        9123     /* EX box port 4 */
#define  cnsUdp3        9133     /* EX box port 4 */

#define  cnsRmtpUdp4    9124     /* EX box port 5 */
#define  cnsTcp4        9124     /* EX box port 5 */
#define  cnsUdp4        9134     /* EX box port 5 */


                  /* SIPX Domain */

#define  cnsRmtpIpx0    TBD_I    /* EX box port 1, Non-EX box, default */
```

```
#define  cnsSpx0        TBD_I    /* EX box port 1, Non-EX box, default */
#define  cnsIpx0        TBD_I    /* EX box port 1, Non-EX box, default */

#define  cnsRmtpIpx1    TBD_I    /* EX box port 2 */
#define  cnsSpx1        TBD_I    /* EX box port 2 */
#define  cnsIpx1        TBD_I    /* EX box port 2 */

#define  cnsRmtpIpx2    TBD_I    /* EX box port 3 */
#define  cnsSpx2        TBD_I    /* EX box port 3 */
#define  cnsIpx2        TBD_I    /* EX box port 3 */

#define  cnsRmtpIpx3    TBD_I    /* EX box port 4 */
#define  cnsSpx3        TBD_I    /* EX box port 4 */
#define  cnsIpx3        TBD_I    /* EX box port 4 */

#define  cnsRmtpIpx4    TBD_I    /* EX box port 5 */
#define  cnsSpx4        TBD_I    /* EX box port 5 */
#define  cnsIpx4        TBD_I    /* EX box port 5 */



/* ================================================================ */
/* |||||||||||||||||| Structures, Types & Classes |||||||||||||||| */
/* ================================================================ */


/*
================================================
A communication domain is an abstraction that implies an addressing
structure and a set of protocols.  For example, the Internet (INET)
domain implies IP addressing along with the TCP and RMTP/UDP
transport protocols.

The following domains are supported if they are installed and active in
the host platform:
   ETdomainINET,        Internet domain:   TCP, RMTP/UDP
   ETdomainSIPX,        Novell domain:     SPX, RMTP/IPX
   ETdomainIrDA,        Infrared domain:   TTP
   ETdomainUnix,        Unix domain:       Unix paths.
   ETdomainLoopBack,    For loopBack testing - no transport.
   ETdomainLAST,        Use this for the number of domains.
   ETdomainNone         Used for initializing
------------------------------------------------ */
typedef  enum
   {
   ETdomainINET,     /* ip:  */
   ETdomainSIPX,     /* ipx: */
   ETdomainIrDA,     /* ir:  */
   ETdomainUnix,     /* up:  */
   ETdomainLoopBack, /* lb:  */
   ETdomainLAST,
   ETdomainNone
   }  Tdomain;
/*
================================================
Addresses in the Internet domain specify an IP address and a port.  The
port is a TCP or a UDP port.

Currently, IP addresses are 32 bits.  They will expand to 128 bits in
the future with IPv6.  This structure will accommodate the new 128 bit
addresses.  (4 bytes to 16 bytes)
------------------------------------------------ */
```

```
typedef  struct   TinetAddr_
   {
   Tu8   mbrPort     [cnsINETPort];    /* 16-bits  */
   Tu8   mbrIPAddr   [cnsINETIPAddr];  /* 128-bits */
   }  TinetAddr;
/*
================================================
Addresses in the Novell domain...
------------------------------------------------ */
typedef  struct   TipxAddr_
   {
   Tu8   mbrNetwork  [cnsSIPXNet];     /* 4-giga networks */
   Tu8   mbrNode     [cnsSIPXNode];
   Tu8   mbrSocket   [cnsSIPXSock];
   }  TipxAddr;
/*
================================================
Addresses in the Infrared domain...
------------------------------------------------ */
typedef  struct   TirdaAddr_
   {
   Tu8   mbrDeviceAddr  [TBD_I];
   Tu8   mbrLsapSel     [TBD_I];
   }  TirdaAddr;
/*
================================================
Addresses in the Unix domain...
------------------------------------------------ */
typedef  struct   TunixAddr_
   {
   Tu8   mbrPath  [cnsUnixPath];
   }  TunixAddr;
/*
================================================
Addresses in the LoopBack domain...
------------------------------------------------ */
typedef  struct   TloopBackAddr_
   {
   Tu8   mbrNothing;
   }  TloopBackAddr;
/*
================================================
TIAPI addresses specify a domain and include a corresponding domain
specific address structure.
------------------------------------------------ */
typedef  union
   {
   TinetAddr      mbrINETaddr;
   TipxAddr       mbrIPXAddr;
   TirdaAddr      mbrIrDAAddr;
   TunixAddr      mbrUnixAddr;
   TloopBackAddr  mbrLoopBackAddr;
   }  Taddress;

typedef  struct   Taddr_
   {
   Tdomain  mbrDomain;
   Taddress mbrAddress;
   }  Taddr;
/*
================================================
TIAPI supports two types of communications service - message and
```

```
stream.

Message service uses the Reliable Message Transport Protocol (RMTP)
running over the datagram protocol (UDP,IPX, etc.) that is available in
the domain.

Stream service uses the standard connection-mode, full-duplex, reliable
transports such as TCP, SPX, etc..
---------------------------------------------- */
typedef  enum
   {
   ETsrvcTypeMessage,
   ETsrvcTypeStream
   }  TsrvcType;
/*
================================================
TIAPI provides an unsolicited reply -- TrplyNwkStat -- that tells OSP
that a local network address is active or that the local network
address has changed to inactive.  This typedef defines those statuses.
See the description section of TrplyNwkStat to find out what is ment by
local addresses becomming active or inactive.

The possible status values that TIAPI will put into a TrplyNwkStat.
   ETnwkStatNotAvail - means that a network address that was
                       previously reported in a TrplyNwkStat as
                       being available is now unavailable.
   ETnwkStatAvail    - means that a local network address is available
                       for OSP to use.
---------------------------------------------- */
typedef  enum
   {
   ETnwkStatNotAvail,
   ETnwkStatAvail
   }  TnwkStat;
/*
================================================
OSP uses a TconnHdl to identify a connection.  TIAPI creates the
TconnHdl when a connection is established and passes it up to the OSP.
From that point on, the OSP uses the TconnHdl to identify to the TIAPI
the connection upon which the TIAPI should perform an operation.

TconnHdl is used as a handle to refer to a connection in all connection
related interactions between TIAPI and OSP.
---------------------------------------------- */
typedef  void* TconnHdl;
/*
================================================
A connection may be either shutdown gracefully or aborted.  If it is
shutdown gracefully, the underlying transport is asked to gracefully
close the connection if it has that capability.  It does not mean that
TIAPI will complete all xptSend() requests before closing the
connection.  Any pending xptSend() requests that are being worked on by
TIAPI when xptHangUp() is called will be cancelled.
---------------------------------------------- */
typedef  enum
   {
   ETshutdownGraceful,
   ETshutdownAbort
   }  Tshutdown;
/*
================================================
All TIAPI functions return a completion code (TcCode) and all replies
```

```
include a TcCode data member.
---------------------------------------------- */
typedef  enum
   {
   ETcCodeSuccess,
   ETcCodeFailure,
   ETcCodeRequestCancelled,
   ETcCodeInvalidOSPAddr,
   ETcCodeInvalidTIAddr,
   ETcCodeUnsupportedDomain,
   ETcCodeInvalidHostName,
   ETcCodeInvalidAddr,
   ETcCodeAddrNotAvail,
   ETcCodeBusy,
   ETcCodeNoAnswer,
   ETcCodeAbort,
   ETcCodeConnClosed,
   ETcCodeConnFailed,
   ETcCodeInvalidConn,
   ETcCodeTooLarge,
   ETcCodeInvalidDgmSvc,
   ETcRequestNotFound,
   ETcCodeListenAlreadyActive,
   ETcCodeInvalidLocalPort
   }  TcCode;
/*
================================================
The TIAPI contains functions and replies.  Functions pass instructions
and data into the TIAPI.  Replies pass data the other way, from TIAPI
to OSP.

Any function that may send or receive data over the network is
implemented as an asynchronous (non-blocking) call.  The results of its
action are returned as a corresponding reply.

These are the names of the replies that are a part of the TIAPI:
---------------------------------------------- */
typedef  enum
   {
   ETrplyIdXlateOSPAddr,
   ETrplyIdXlateTIAddr,
   ETrplyIdNwkStat,
   ETrplyIdActiveConnect,
   ETrplyIdPassiveConnect,
   ETrplyIdDisconnect,
   ETrplyIdSend,
   ETrplyIdReceive,
   ETrplyIdSendDatagram,
   ETrplyIdReceiveDatagram,
   ETrplyIdError
   }  TrplyId;
/*
================================================
Any function for which there is an asynchronous reply includes a
request handle parameter.  The handle can be used by OSP to
match-up replies with their corresponding requests.  The handle is a
value set by OSP before it calls the request function.  The same handle
value is returned to OSP in the reply that corresponds to that request
call.  Some requests - such as xptListen() - may generate multiple
replies, one for each connection that results from the request.  In
those cases, the request handle from the original request call is
returned in each reply.
```

```
Some replies may not be the direct result of a prior request.  Those
replies will have a request handle of 0.

This request handle is used by TIAPI for identifying previously made
requests.
---------------------------------------------- */
typedef  Ts32  TrqstHdl;



/* ================================================================ */
/* |||||||||||||||| Prototypes, Replies and Events ||||||||||||||| */
/* ================================================================ */


/*
====================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
====================================================================

Name:
   xptInitialize()

Description:
   Initialize or startup TIAPI.

   This must be called before any other TIAPI functions are called.
   It can be called only one time, when TIAPI is started.

Parameters:
   none

Returns:
   ETcCodeSuccess
      if initialization completed successfully.
   ETcCodeFailure
      if TIAPI cannot initialize properly.  If this is returned, TIAPI
      will not function.

--------------------------------------------------------------------
*/
TcCode  xptInitialize
   (
   void
   );



/*
====================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
====================================================================

Name:
   xptShutdown()

Description:
   Shutdown TIAPI.

   When this call is made, all pending requests are canceled by
   TIAPI.  All open connections are aborted.  No replies are generated
```

```
   for the canceled requests.  The OSP layer should assume that all of
   its outstanding TIAPI requests are canceled when this function
   returns.  TIAPI will retrun no more replies through xptGetNextEvent()
   and will not make any further calls into OSP after xptShutdown() is
   called.

   This should be called only one time, when TIAPI is terminated.
   After this call TIAPI will not function again until a
   xptInitialize() call is completed.

Parameters:
   none

Returns:
   none

-----------------------------------------------------------------
*/
void  xptShutdown
   (
   void
   );



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   xptCancelRequest()

Description:
   Cancel a previously issued request.

   TIAPI will cancel all pending requests that have the specified
   prmRqstHdl value.

Parameters:
   prmRqstHdl:
      The request handle that was passed into TIAPI by OSP in the
      function call that made the request(s) that OSP wants to cancel.

Returns:
   ETcCodeSuccess:
      If the request was successfully canceled.
   ETcRequestNotFound
      TIAPI cannot find the specified request.
   ETcCodeFailure:
      The request was found but cannot be canceled.  This code may
      also be returned by a TIAPI implementation that does not
      support cancelling the previously issued request.  (Some
      implementations may only cancel xptListen() requests.)

-----------------------------------------------------------------
*/
TcCode   xptCancelRequest
   (
   TrqstHdl prmRqstHdl
   );
```

```
/*
================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
================================================================

Name:
   xptRun()

Description:
   Accept the thread of control from the caller.

   TOPS is single threaded for portability reasons.  One way TIAPI
   gets hold of the thread is through this call.  When TIAPI gets the
   thread it can use it for whatever processing it needs to do.
   Typically, TIAPI would respond to this call by checking for events
   from the underlying transports.

   When TIAPI has nothing to do, it returns the thread by returning
   from this function.

Parameters:
   none

Returns:
   none

----------------------------------------------------------------
*/
void  xptRun
   (
   void
   );



/*
================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
================================================================

Name:
   xptXlateOSPAddr()

Description:
   Xlate an address that is in OMN form to an address that is in TIAPI
   form.

   OMN form addresses are defined in the HP-JetSend Network ARS
   document.

   - This function may send or receive data over the network and so -
   - is implemented as an asynchronous (non-blocking) call.  The    -
   - results of its action are returned as a corresponding reply.   -

   If the OMN form address contains a hostName, xptXlateOSPAddr() uses
   the appropriate Name Service to translate the address.
      DNS, WINS, NIS, X.500, SAP, etc.

Parameters:
   prmRqstHdl:
```

```
      An OSP generated request handle that TIAPI will include in its
      reply to OSP that corresponds to this call.  OSP can use the
      handle to match-up its calls to asynchronous functions with the
      replies it gets back for those functions.
   prmOSPAddrPtr:
      Pointer to a zero terminated ASCII string containing the OMN
      form address to translate.

Returns:
   ETcCodeSuccess:
      If the request was successfully enqueued by TIAPI.  TIAPI will
      act upon the request and generate an asynchronous reply to it.
   ETcCodeFailure:
      TIAPI could not enqueue the request.  This may happen if TIAPI
      is out of queue resources.

-----------------------------------------------------------------
*/
TcCode   xptXlateOSPAddr
   (
   TrqstHdl prmRqstHdl,
   Tchar*   prmOSPAddrPtr
   );



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   typedef TrplyXlateOSPAddr

Description:
   The reply that corresponds to and returns results from the call to
   xptXlateOSPAddr().

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdXlateOSPAddr
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            TIAPI wrote a valid address translation into
            mbrTIAddr.
         ETcCodeRequestCancelled:
            This request was canceled with a call to
            xptCancelRequest().
         ETcCodeInvalidOSPAddr:
            If *prmOSPAddrPtr contains a syntactically invalid
            address.
         ETcCodeUnsupportedDomain:
            If *prmOSPAddrPtr cannot be translated because the
            domain it specifies is not supported.
         ETcCodeInvalidHostName:
            If *prmOSPAddrPtr cannot be translated because the host
            name it specifies cannot be resolved into a network
            address.
   mbrRqstHdl:
```

```
      The request handle that was passed into TIAPI by OSP in the
      function call that corresponds to this reply.
   mbrTIAddrPtr:
      The structure into which the address translation is written.

-------------------------------------------------------------
*/
typedef  struct   TrplyXlateOSPAddr_
   {
   TrplyId  mbrRplyId;
   TcCode   mbrCCode;
   TrqstHdl mbrRqstHdl;
   Taddr    mbrTIAddr;
   Tchar*   mbrOSPAddrPtr;
   }  TrplyXlateOSPAddr;



/*
=============================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=============================================================

Name:
   xptXlateTIAddr()

Description:
   Xlate an address that is in TIAPI form to an address that is in OMN
   form.

   OMN form addresses are defined in the HP-JetSend Network ARS
   document.

   - This function may send or receive data over the network and so -
   - is implemented as an asynchronous (non-blocking) call.  The    -
   - results of its action are returned as a corresponding reply.   -

Parameters:
   prmRqstHdl:
      An OSP generated request handle that TIAPI will include in its
      reply to OSP that corresponds to this call.  OSP can use the
      handle to match-up its calls to asynchronous functions with the
      replies it gets back for those functions.
   prmTIAddrPtr:
      Pointer to a Taddr containing the TIAPI address to translate.
   prmMaxBytes:
      The maximum number of bytes that can be written into
      *prmOSPAddrPtr.
   prmOSPAddrPtr:
      A pointer to a buffer into which the adress translation will
      be written.

Returns:
   ETcCodeSuccess:
      If the request was successfully enqueued by TIAPI.  TIAPI will
      act upon the request and generate an asynchronous reply to it.
   ETcCodeFailure:
      TIAPI could not enqueue the request.  This may happen if TIAPI
      is out of queue resources.

-------------------------------------------------------------
*/
```

```
TcCode   xptXlateTIAddr
   (
   TrqstHdl prmRqstHdl,
   Taddr*   prmTIAddrPtr,
   Ts32     prmMaxBytes,
   Tchar*   prmOSPAddrPtr
   );



/*
================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
================================================================

Name:
   typedef TrplyXlateTIAddr

Description:
   The reply that corresponds to and returns results from the call to
   xptXlateTIAddr().

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdXlateTIAddr
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            TIAPI wrote a valid address translation into
            *mbrOSPAddrPtr.
         ETcCodeRequestCancelled:
            This request was canceled with a call to
            xptCancelRequest().
         ETcCodeInvalidTIAddr:
            If *prmTIAddrPtr contains an invalid address.
         ETcCodeUnsupportedDomain:
            If *prmTIAddrPtr cannot be translated because the
            domain it specifies is not supported.
   mbrRqstHdl:
      The request handle that was passed into TIAPI by TI in the
      function call that corresponds to this reply.
   mbrOSPAddrPtr:
      The same Tchar* that was passed into TIAPI via the corresponding
      call to xptXlateTIAddr().
      A pointer to a buffer containing the translated address.  The
      OMN form address is stored as a zero trminated ASCII string.

----------------------------------------------------------------
*/
typedef  struct   TrplyXlateTIAddr_
   {
   TrplyId  mbrRplyId;
   TcCode   mbrCCode;
   TrqstHdl mbrRqstHdl;
   Taddr*   mbrTIAddrPtr;
   Tchar*   mbrOSPAddrPtr;
   }  TrplyXlateTIAddr;
```

```
/*
================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
================================================================

Name:
   typedef TrplyNwkStat

Description:
   This is an unsolicited reply that TIAPI generates anytime the
   status of one of the underlying networks changes.  When TIAPI first
   detects that a local address is available it sends a TrplyNwkStatus
   to OSP notifying OSP that it can use that address to establish
   connections.  Likewise, when TIAPI detects that an address is no
   longer available it notifies OSP of that event also.

   A single HP-JetSend device may be located at multiple network addresses.
   It may be located at addresses in multiple domains as well as at
   multiple addresses within a single domain.  As an example, a HP-JetSend
   device within a printer containing two JetDirect cards could be
   found at all of the following addresses simultaneously:
      JD card 1:
         SIPX address:  12 34 56 78 90 AB 12 34 56 78
         INET address:  15.62.94.23
      JD card 2:
         INET address:  15.39.20.45
   One card has IPX and IP enabled while the second card has only IP
   enabled.

   These address may change while HP-JetSend is running.  Using the
   example above, the network administrator may do any of the
   following while HP-JetSend is running:
      - Disable one or more of the addresses.
      - Enable one or more new addresses (such as an IPX address on
        the second card.
      - Change one or more of the addresses.

   In the example above, if all three addresses are initially active
   when HP-JetSend starts running, TIAPI will generate three TrplyNwkStatus
   events for OSP - one for each address.  If at a later time the
   administrator disables INET address 15.39.20.45, TIAPI will
   send a new TrplyNwkStatus for that address to OSP telling it that
   the address is no longer available.  If the administrator changes
   INET address 15.62.94.23 to 15.62.89.67, TIAPI will send a
   TrplyNwkStatus for 15.62.94.23 telling OSP it is no longer
   available.  Then TIAPI will send another TrplyNwkStatus, this one
   for 15.62.89.67, telling OSP it is available.

   If OSP has connections open through an address when the address is
   disabled or changed, OSP does not have to take any special action.
   Those connections will simply fail and OSP will receive a
   TrplyDisconnect just as it would if the connection failed for any
   other reason.

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdNwkStat
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
```

```
   mbrRqstHdl:
      No meaning, always set to 0.
   mbrLocalAddress:
      The network domain and network address about which TIAPI is
      returning status.
         The mbrPort field (or mbrSocket field in SIPX addresses) is
         set to the well-known RMTP port within the domain.  If the
         HP-JetSend appliance is attached to one of the parallel ports
         of a multi-port EX box, TIAPI will compute the correct
         well-known RMTP port for that parallel port.  See the
         HP-JetSend Network ARS section on Multi-Port Network-Parallel
         Gateways.
   mbrStatus:
      The status - see typedef TnwkStat for possible values.

-----------------------------------------------------------------
*/
typedef  struct   TrplyNwkStat_
   {
   TrplyId  mbrRplyId;
   TcCode   mbrCCode;
   TrqstHdl mbrRqstHdl;
   Taddr    mbrLocalAddr;
   TnwkStat mbrStatus;
   }  TrplyNwkStat;



/*
==================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
==================================================================

Name:
   xptSendDatagram()

Description:
   Send a buffer of data through as a datagram.

   - This function may send or receive data over the network and so -
   - is implemented as an asynchronous (non-blocking) call.  The    -
   - results of its action are returned as a corresponding reply.   -

Parameters:
   prmRqstHdl:
      An OSP generated request handle that TIAPI will include in its
      reply to OSP that corresponds to this call.  OSP can use the
      handle to match-up its calls to asynchronous functions with the
      replies it gets back for those functions.
   prmLocalAddrPtr:
      A pointer to a TIAPI address structure containing the address
      of the local host through which TIAPI should attempt to
      send the datagram.  If prmLocalAddressPtr is NULL, TIAPI will
      select the local address through which it will attempt to send
      the datagram (same behavior as using INADDR_ANY in
      BerkeleySockets).  If it is not NULL, TIAPI will use the local
      address it points to when attempting to send the datagram.
         - If the port field (or socket field in the case of SIPX
           domain) in the address is zero, TIAPI will send through
           its well-known datagram port.  If prmLocalAddrPtr is
           NULL, the port field is treated as if it were set to
           zero.
```

```
            - else if the port field contains one of the well-known
              RMTP port numbers, TIAPI will compute the corresponding
              well-known datagram port and send through that port.
            - Otherwise, TIAPI will open the specified port and send
              through it.
      prmForeignAddrPtr:
         A pointer to a TIAPI address structure containing the address
         of the foreign peer with which TIAPI should attempt to
         send the datagram.
            - If the port field (or socket field in the case of SIPX
              domain) in the address is zero, TIAPI will send to the
              default well-known datagram port number.
            - else if the port field contains one of the well-known
              RMTP port numbers, TIAPI will compute the corresponding
              well-known datagram port and send to that port.
            - Otherwise, TIAPI will send to the specified port.
      prmDataPtr:
         A pointer to a buffer control block that xptSendDatagram()
         will use to send data.  xptSendDatagram() will assume
         that the payload portion of the Tbcb is large enough
         to send prmMaxBytes of data from.  TIAPI will
         simply send data from the buffer at the address returned
         by calling BufDataAddr() on the Tbcb.
      prmBytesToSend:
         The length of the data pointed to by prmDataPtr.

   Returns:
      ETcCodeSuccess:
         If the request was successfully enqueued by TIAPI.  TIAPI will
         act upon the request and generate an asynchronous reply to it.
      ETcCodeFailure:
         TIAPI could not enqueue the request.  This may happen if TIAPI
         is out of queue resources.

   ------------------------------------------------------------------
   */
   TcCode   xptSendDatagram
      (
      TrqstHdl prmRqstHdl,
      Taddr*   prmLocalAddrPtr,
      Taddr*   prmForeignAddrPtr,
      Tbcb*    prmDataPtr,
      Ts32     prmBytesToSend
      );



   /*
   ====================================================================
   [][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
   ====================================================================

   Name:
      typedef TrplySendDatagram

   Description:
      The reply that corresponds to and returns results from the call to
      xptSendDatagram().

   Fields:
      mbrRplyId:
         Identifies to the receiver which kind of reply this is. Always
```

```
      equal to:
         ETrplyIdSendDatagram
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            The data was sent.
         ETcCodeFailure:
            Internal resources are unavailable to complete the
            request.
         ETcCodeRequestCancelled:
            This request was canceled with a call to
            xptCancelRequest().
         ETcCodeTooLarge:
            The data is larger than the underlying transport will
            support.
         ETcCodeInvalidLocalPort:
            TIAPI cannot use the specified local port for this
            request.
   mbrRqstHdl:
      The request handle that was passed into TIAPI by OSP in the
      function call that corresponds to this reply.
   mbrLocalAddr:
      A structure containing the local address through which TIAPI
      sent the datagram.
   mbrForeignAddr:
      A structure containing the address of the foreign peer to
      which TIAPI sent the datagram.
   mbrDataPtr:
      The same Tbcb* that was passed into TIAPI via the corresponding
      call to xptSendDatagram().
      A pointer to a buffer containing the data that was transmitted.
   mbrBytesSent:
      The number of bytes actually transmitted.

-------------------------------------------------------------------
*/
typedef  struct   TrplySendDatagram_
   {
   TrplyId  mbrRplyId;
   TcCode   mbrCCode;
   TrqstHdl mbrRqstHdl;
   Taddr    mbrLocalAddr;
   Taddr    mbrForeignAddr;
   Tbcb*    mbrDataPtr;
   Ts32     mbrBytesSent;
   }  TrplySendDatagram;



/*
====================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
====================================================================

Name:
   xptReceiveDatagram()

Description:
   Accept a buffer of data from a datagram.
   This function passes a buffer into TIAPI that TIAPI can use for
   receiving a datagram.
```

```
   - This function may send or receive data over the network and so -
   - is implemented as an asynchronous (non-blocking) call.  The    -
   - results of its action are returned as a corresponding reply.   -

Parameters:
   prmRqstHdl:
      An OSP generated request handle that TIAPI will include in its
      reply to OSP that corresponds to this call.  OSP can use the
      handle to match-up its calls to asynchronous functions with the
      replies it gets back for those functions.
   prmLocalAddrPtr:
      A pointer to a TIAPI address structure containing the address
      of the local host over which TIAPI should listen for a datagram.
         - If the port field (or socket field in the case of SIPX
           domain) in the address is zero, TIAPI will listen through
           its well-known datagram port.
         - else if the port field contains one of the well-known
           RMTP port numbers, TIAPI will compute the corresponding
           well-known datagram port and listen through that port.
         - Otherwise, TIAPI will open the specified port and listen
           through it.
   prmMaxBytes:
      The maximum number of bytes that can be written into
      *prmDataPtr.
   prmDataPtr:
      A pointer to a buffer control block that xptReceiveDatagram()
      will use to receive data.  xptReceiveDatagram() will assume
      that the payload portion of the Tbcb is large enough
      to receive prmMaxBytes of data.  Prior to calling
      xptReceiveDatagram(), OSP will have called PrependBuf() to
      make room in the Tbcb for prmMaxBytes of data.  TIAPI will
      simply receive data into the buffer at the address returned
      by calling BufDataAddr() on the Tbcb.

Returns:
   ETcCodeSuccess:
      If the request was successfully enqueued by TIAPI.  TIAPI will
      act upon the request and generate an asynchronous reply to it.
   ETcCodeFailure:
      TIAPI could not enqueue the request.  This may happen if TIAPI
      is out of queue resources.

-------------------------------------------------------------------
*/
TcCode   xptReceiveDatagram
   (
   TrqstHdl prmRqstHdl,
   Taddr*   prmLocalAddrPtr,
   Ts32     prmMaxBytes,
   Tbcb*    prmDataPtr
   );



/*
====================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
====================================================================

Name:
   typedef TrplyReceiveDatagram
```

```
Description:
   The reply that corresponds to and returns results from the call to
   xptReceiveDatagram().

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdReceiveDatagram
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            Data was received.
         ETcCodeFailure:
            Internal resources are unavailable to complete the
            request.
         ETcCodeRequestCancelled:
            This request was canceled with a call to
            xptCancelRequest().
         ETcCodeInvalidLocalPort:
            TIAPI cannot use the specified local port for this
            request.
   mbrRqstHdl:
      The request handle that was passed into TIAPI by OSP in the
      function call that corresponds to this reply.
   mbrLocalAddr:
      A structure containing the local address through which TIAPI
      received the datagram.
   mbrForeignAddr:
      A structure containing the address of the foreign peer from
      which TIAPI has received a datagram.
   mbrDataPtr:
      The same Tbcb* that was passed into TIAPI via the corresponding
      call to xptReceiveDatagram().
      A pointer to a buffer containing the data that was received.
   mbrBytesReceived:
      The number of bytes received.

-------------------------------------------------------------------
*/
typedef  struct   TrplyReceiveDatagram_
   {
   TrplyId  mbrRplyId;
   TcCode   mbrCCode;
   TrqstHdl mbrRqstHdl;
   Taddr    mbrLocalAddr;
   Taddr    mbrForeignAddr;
   Tbcb*    mbrDataPtr;
   Ts32     mbrBytesReceived;
   }  TrplyReceiveDatagram;



/*
===================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
===================================================================

Name:
   xptCall()

Description:
```

Attempt to open a connection to another device.

- This function may send or receive data over the network and so -
- is implemented as an asynchronous (non-blocking) call. The -
- results of its action are returned as a corresponding reply. -

A platform that is multi-homed is one that has more than one local address within the same domain. For example, a LaserJet with 2 JetDirect cards may have 2 IP addresses. In some cases it may be possible to connect to a foreign peer through one of those local addresses but not the other. If prmLocalAddrPtr is NULL, xptCall() will select the local address through which it can establish a connection to the foreign peer -- if any.

Parameters:

prmRqstHdl:

An OSP generated request handle that TIAPI will include in its reply to OSP that corresponds to this call. OSP can use the handle to match-up its calls to asynchronous functions with the replies it gets back for those functions.

prmServiceType:

Indicates the type of connection desired - STREAM or MESSAGE.

prmLocalAddrPtr:

A pointer to a TIAPI address structure containing the address of the local host through which TIAPI should attempt to establish a connection. If prmLocalAddressPtr is NULL, TIAPI will select the local address through which it will attempt to establish the connection (same behavior as using INADDR_ANY in BerkeleySockets). If it is not NULL, TIAPI will use the local address it points to when attempting to establish the connection.

* if the request is for a MESSAGE connection and...
  - If the port field (or socket field in the case of SIPX domain) in the address is zero, TIAPI will call through its well-known RMTP port. If prmLocalAddrPtr is NULL, the port field is treated as if it were set to zero.
  - Otherwise, TIAPI will open the specified port and call through it.
* if the request is for a STREAM connection and...
  - If the port field (or socket field in the case of SIPX domain) in the address is zero, TIAPI will call through its well-known reliable-connection-oriented port. If prmLocalAddrPtr is NULL, the port field is treated as if it were set to zero.
  - else if the port field contains one of the well-known RMTP port numbers, TIAPI will compute the corresponding well-known reliable-connection-oriented port and call through that port.
  - Otherwise, TIAPI will open the specified port and call through it.

prmForeignAddrPtr:

A pointer to a TIAPI address structure containing the address of the foreign peer with which TIAPI should attempt to establish a connection.

* if the request is for a MESSAGE connection and...
  - If the port field (or socket field in the case of SIPX domain) in the address is zero, TIAPI will call to the well-known RMTP port.
  - Otherwise, TIAPI will call to the specified port.
* if the request is for a STREAM connection and...
  - If the port field (or socket field in the case of SIPX

```
                domain) in the address is zero, TIAPI will call to
                the well-known reliable-connection-oriented port.
              - else if the port field contains one of the well-known
                RMTP port numbers, TIAPI will compute the corresponding
                well-known reliable-connection-oriented  port and
                call to that port.
              - Otherwise, TIAPI will call to the specified port.

Returns:
   ETcCodeSuccess:
      If the request was successfully enqueued by TIAPI.  TIAPI will
      act upon the request and generate an asynchronous reply to it.
   ETcCodeFailure:
      TIAPI could not enqueue the request.  This may happen if TIAPI
      is out of queue resources.

Semantics:
   lclSkt  = socket(prmLocalAddrPtr->mbrDomain,STREAM,0);
   bind(lclSkt,&prmLocalAddrPtr->mbrAddress.mbrINETAddr,lclAddrSize);
   connect
      (
      lclSkt,
      &lclRply->mbrForeignAddr->mbrAddress.mbrINETAddr,
      &lclAddrSize
      );

-----------------------------------------------------------------
*/
TcCode   xptCall
   (
   TrqstHdl    prmRqstHdl,
   TsrvcType   prmServiceType,
   Taddr*      prmLocalAddrPtr,
   Taddr*      prmForeignAddrPtr
   );



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   typedef TrplyActiveConnect

Description:
   The reply that corresponds to and returns results from the call to
   xptCall().

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdActiveConnect
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            A connection is established and ready to use.
         ETcCodeFailure:
            Internal resources are unavailable to complete the
            request.
```

```
            ETcCodeRequestCancelled:
               This request was canceled with a call to
               xptCancelRequest().
            ETcCodeUnsupportedDomain:
               The domain specified in the address is not supported.
            ETcCodeInvalidAddr:
               The specified address is syntactically invalid.
            ETcCodeAddrNotAvail:
               The specified address is not available from the local
               host.
            ETcCodeBusy:
               The attempt to connect was refused.
            ETcCodeNoAnswer:
               The attempt to connect timed-out.
            ETcCodeInvalidLocalPort:
               TIAPI cannot use the specified local port for this
               request.
      mbrRqstHdl:
         The request handle that was passed into TIAPI by OSP in the
         function call that corresponds to this reply.
      mbrLocalAddr:
         A structure containing the local address through which TIAPI
         has established a connection.
      mbrForeignAddr:
         A structure containing the address of the foreign peer with
         which TIAPI has established a connection.
      mbrConnHdl:
         The identifier that OSP will use in all its interactions with
         TIAPI to refer to this connection.
      mbrMaxMsgSize:
         The maximum size of a single OSP message (including the OSP
         header) that the transport can send on this connection.

   ------------------------------------------------------------------
   */
   typedef  struct   TrplyActiveConnect_
      {
      TrplyId  mbrRplyId;
      TcCode   mbrCCode;
      TrqstHdl mbrRqstHdl;
      Taddr    mbrLocalAddr;
      Taddr    mbrForeignAddr;
      TconnHdl mbrConnHdl;
      Tu32     mbrMaxMsgSize;
      }  TrplyActiveConnect;



   /*
   ====================================================================
   [][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
   ====================================================================

   Name:
      xptListen()

   Description:
      Accept connection requests from other devices.

      - This function may send or receive data over the network and so -
      - is implemented as an asynchronous (non-blocking) call.  The    -
      - results of its action are returned as a corresponding reply.   -
```

Parameters:

prmRqstHdl:

An OSP generated request handle that TIAPI will include in its reply to OSP that corresponds to this call. OSP can use the handle to match-up its calls to asynchronous functions with the replies it gets back for those functions.

prmServiceType:

Indicates the type of connections to listen for - STREAM or MESSAGE.

prmLocalAddrPtr:

A pointer to a TIAPI address structure containing the address of the local host over which TIAPI should listen for connection attempts.

* if the request is for a MESSAGE connection and...
  - If the port field (or socket field in the case of SIPX domain) in the address is zero, TIAPI will listen through its well-known RMTP port.
  - Otherwise, TIAPI will open the specified port and listen through it.
* if the request is for a STREAM connection and...
  - If the port field (or socket field in the case of SIPX domain) in the address is zero, TIAPI will listen through its well-known reliable-connection-oriented port.
  - else if the port field contains one of the well-known RMTP port numbers, TIAPI will compute the corresponding well-known reliable-connection-oriented port and listen through that port.
  - Otherwise, TIAPI will open the specified port and listen through it.

Returns:

ETcCodeSuccess:

If the request was successfully enqueued by TIAPI. TIAPI will act upon the request and generate asynchronous replies to it.

ETcCodeFailure:

TIAPI could not enqueue the request. This may happen if TIAPI is out of queue resources.

Semantics:

```
lclSkt  = socket(prmLocalAddrPtr->mbrDomain,STREAM,0);
bind(lclSkt,&prmLocalAddrPtr->mbrAddress.mbrINETAddr,lclAddrSize);
listen(lclSkt,2);
while (1)
   {
   lclRply = malloc(sizeof(TrplyPassiveConnect));
   lclRply->mbrRplyId  = ETrplyIdPassiveConnect;
   lclRply->mbrRqstHdl = prmRqstHdl;
   lclSize   = sizeof(lclRply->mbrForeignAddr);
   lclNewSkt = accept
      (
      lclSkt,
      &lclRply->mbrForeignAddr,
      &lclSize
      );
   lclRply->mbrConnHdl = //a unique connection identifier//
   lclRply->mbrCCode   = ////
   //put lclRply on queue from which OSP retrieves events //
   //and replies.//
   }
```

```
-----------------------------------------------------------------
*/
TcCode   xptListen
   (
   TrqstHdl    prmRqstHdl,
   TsrvcType   prmServiceType,
   Taddr*      prmLocalAddrPtr
   );



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   typedef TrplyPassiveConnect

Description:
   The reply that corresponds to and returns results from the call to
   xptListen().

   For reasons of consistency of usage and to cede to OSP control over
   when and whether connections within a particular domain (like IrDA)
   are accepted, OSP must have called xptListen() on the domain before
   a connection can be accepted in that domain.

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdPassiveConnect
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            A connection is established and ready to use.
         ETcCodeFailure:
            Internal resources are unavailable to complete the
            request.
         ETcCodeRequestCancelled:
            This request was canceled with a call to
            xptCancelRequest().
         ETcCodeUnsupportedDomain:
            The domain specified in the address is not supported.
         ETcCodeInvalidAddr:
            The specified address is syntactically invalid.
         ETcCodeAddrNotAvail:
            The specified address is not available from the local
            host.
         ETcCodeListenAlreadyActive:
            TIAPI is already listening for connection requests on
            the specified local address.
         ETcCodeInvalidLocalPort:
            TIAPI cannot use the specified local port for this
            request.
   mbrRqstHdl:
      The request handle that was passed into TIAPI by OSP in the
      function call that corresponds to this reply.
   mbrLocalAddr:
      A structure containing the local address through which TIAPI
      has established a connection.
```

```
   mbrForeignAddr:
      A structure containing the address of the foreign peer with
      which TIAPI has established a connection.
   mbrConnHdl:
      The identifier that OSP will use in all its interactions with
      TIAPI to refer to this connection.
   mbrMaxMsgSize:
      The maximum size of a single OSP message (including the OSP
      header) that the transport can send on this connection.

-----------------------------------------------------------------
*/
typedef  struct   TrplyPassiveConnect_
   {
   TrplyId  mbrRplyId;
   TcCode   mbrCCode;
   TrqstHdl mbrRqstHdl;
   Taddr    mbrLocalAddr;
   Taddr    mbrForeignAddr;
   TconnHdl mbrConnHdl;
   Tu32     mbrMaxMsgSize;
   }  TrplyPassiveConnect;



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   xptHangUp()

Description:
   Close a connection to another device.

   * This request cannot be canceled. *

   - This function may send or receive data over the network and so -
   - is implemented as an asynchronous (non-blocking) call.  The    -
   - results of its action are returned as a corresponding reply.   -

Parameters:
   prmRqstHdl:
      An OSP generated request handle that TIAPI will include in its
      reply to OSP that corresponds to this call.  OSP can use the
      handle to match-up its calls to asynchronous functions with the
      replies it gets back for those functions.
   prmConnHdl:
      Identifier, previously generated and returned by TIAPI, that
      tells TIAPI the connection upon which it should perform this
      operation.
   prmHow:
      Selects whether the connection should terminate gracefully or
      be aborted immediately.

Returns:
   ETcCodeSuccess:
      If the request was successfully enqueued by TIAPI.  TIAPI will
      act upon the request and generate an asynchronous reply to it.
   ETcCodeFailure:
      TIAPI could not enqueue the request.  This may happen if TIAPI
```

```
      is out of queue resources.
   ETcCodeInvalidConn:
      The specified connection handle is invalid.

Semantics:
   lclSkt = //socket associated with connection//
   shutdown(lclSkt,prmHow);
   close(lclSkt);

-----------------------------------------------------------------
*/
TcCode   xptHangUp
   (
   TrqstHdl    prmRqstHdl,
   TconnHdl    prmConnHdl,
   Tshutdown   prmHow
   );



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   typedef TrplyDisconnect

Description:
   The reply that corresponds to and returns results from the call to
   xptHangUp().  It may also be generated in response to a connection
   failure or a request by the foreign peer to hang up.  (OSP must
   always be prepared to receive one of these replies on an open
   connection even if OSP has not issued a xptHangUp() call.)

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdDisconnect
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            The connection closed as gracefully as the underlying
            transport allows.  The connection is no longer usable.
         ETcCodeAbort:
            The connection closed less gracefully than the
            underlying transport allows.  The connection is no
            longer usable.
         ETcCodeInvalidConn:
            The specified connection handle is invalid.
         ETcCodeConnClosed:
            The connection was closed by the foreign peer.
            The connection is no longer usable.
         ETcCodeConnFailed:
            The connection failed due to a time out or other
            failure.  The connection is no longer usable.
   mbrRqstHdl:
      The request handle that was passed into TIAPI by OSP in the
      function call that corresponds to this reply.  The value is 0
      if this reply was generated in response to a connection
      failure or a request by the foreign peer to hang up (if the
```

```
      value of mbrCCode is ETcCodeConnFailed or ETcCodeConnClosed).
   mbrConnHdl:
      Identifies which connection was closed.

-------------------------------------------------------------------
*/
typedef  struct   TrplyDisconnect_
   {
   TrplyId  mbrRplyId;
   TcCode   mbrCCode;
   TrqstHdl mbrRqstHdl;
   TconnHdl mbrConnHdl;
   }  TrplyDisconnect;



/*
====================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
====================================================================

Name:
   xptSend()

Description:
   Send a buffer of data through a connection to another device.

   - This function may send or receive data over the network and so -
   - is implemented as an asynchronous (non-blocking) call.  The    -
   - results of its action are returned as a corresponding reply.   -

Parameters:
   prmRqstHdl:
      An OSP generated request handle that TIAPI will include in its
      reply to OSP that corresponds to this call.  OSP can use the
      handle to match-up its calls to asynchronous functions with the
      replies it gets back for those functions.
   prmConnHdl:
      Identifier, previously generated and returned by TIAPI, that
      tells TIAPI the connection upon which it should perform this
      operation.
   prmDataPtr:
      A pointer to a buffer control block that xptSend()
      will use to send data.  xptSend() will assume
      that the payload portion of the Tbcb is large enough
      to send prmMaxBytes of data from.  TIAPI will
      simply send data from the buffer at the address returned
      by calling BufDataAddr() on the Tbcb.
   prmBytesToSend:
      The length of the data pointed to by prmDataPtr.

Returns:
   ETcCodeSuccess:
      If the request was successfully enqueued by TIAPI.  TIAPI will
      act upon the request and generate an asynchronous reply to it.
   ETcCodeFailure:
      TIAPI could not enqueue the request.  This may happen if TIAPI
      is out of queue resources.
   ETcCodeInvalidConn:
      The specified connection handle is invalid.

Semantics:
```

```
   lclSkt = //socket associated with connection//
   write
      (
      lclSkt,
      BufDataAddr(prmDataPtr),
      prmBytesToSend
      );

-----------------------------------------------------------------
*/
TcCode   xptSend
   (
   TrqstHdl prmRqstHdl,
   TconnHdl prmConnHdl,
   Tbcb*    prmDataPtr,
   Ts32     prmBytesToSend
   );



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   typedef TrplySend

Description:
   The reply that corresponds to and returns results from the call to
   xptSend().

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdSend
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            The data was sent.
         ETcCodeFailure:
            Internal resources are unavailable to complete the
            request.  This may mean the connection has failed.  In
            that case TIAPI will generate a TrplyDisconnect for the
            connection.
         ETcCodeRequestCancelled:
            This request was canceled with a call to
            xptCancelRequest().
         ETcCodeTooLarge:
            The data is larger than the underlying transport will
            support.
         ETcCodeInvalidConn:
            The specified connection handle is invalid.
   mbrRqstHdl:
      The request handle that was passed into TIAPI by OSP in the
      function call that corresponds to this reply.
   mbrConnHdl:
      Identifies which connection data was transmitted over.
   mbrDataPtr:
      The same Tbcb* that was passed into TIAPI via the corresponding
      call to xptSend().
```

```
      A pointer to a buffer containing the data that was transmitted.
   mbrBytesSent:
      The number of bytes actually transmitted.

-----------------------------------------------------------------
*/
typedef  struct   TrplySend_
   {
   TrplyId  mbrRplyId;
   TcCode   mbrCCode;
   TrqstHdl mbrRqstHdl;
   TconnHdl mbrConnHdl;
   Tbcb*    mbrDataPtr;
   Ts32     mbrBytesSent;
   }  TrplySend;



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   xptReceive()

Description:
   Accept a buffer of data through a connection to another device.
   This function passes buffers into TIAPI that TIAPI can use for
   receiving data over a connection.

   - This function may send or receive data over the network and so -
   - is implemented as an asynchronous (non-blocking) call.  The    -
   - results of its action are returned as a corresponding reply.   -

Parameters:
   prmRqstHdl:
      An OSP generated request handle that TIAPI will include in its
      reply to OSP that corresponds to this call.  OSP can use the
      handle to match-up its calls to asynchronous functions with the
      replies it gets back for those functions.
   prmConnHdl:
      Identifier, previously generated and returned by TIAPI, that
      tells TIAPI the connection upon which it should perform this
      operation.
   prmMaxBytes:
      The maximum number of bytes that can be written into
      *prmDataPtr.
   prmDataPtr:
      A pointer to a buffer control block that xptReceive()
      will use to receive data.  xptReceive() will assume
      that the payload portion of the Tbcb is large enough
      to receive prmMaxBytes of data.  Prior to calling
      xptReceive(), OSP will have called PrependBuf() to
      make room in the Tbcb for prmMaxBytes of data.  TIAPI will
      simply receive data into the buffer at the address returned
      by calling BufDataAddr() on the Tbcb.  If it is a request to
      receive from a MESSAGE connection (RMTP), OSP must prepend
      additional space in the buffer to accomadate the RMTP header.

Returns:
   ETcCodeSuccess:
```

```
      If the request was successfully enqueued by TIAPI.  TIAPI will
      act upon the request and generate an asynchronous reply to it.
   ETcCodeFailure:
      TIAPI could not enqueue the request.  This may happen if TIAPI
      is out of queue resources.
   ETcCodeInvalidConn:
      The specified connection handle is invalid.

Semantics:
   OSP will call xptReceive() with code similar to this:
      lclDataPtr = CreateBuf(lclMaxBytes);
      PrependBuf(lclDataPtr,lclMaxBytes);
      xptReceive(lclRqstHdl,lclConnHdl,lclMaxBytes,lclDataPtr);

   and xptReceive will process the call with code similar to this:
      lclSkt = //socket associated with connection//
      read
         (
         lclSkt,
         BufDataAddr(prmDataPtr),
         prmMaxBytes
         );

-----------------------------------------------------------------
*/
TcCode   xptReceive
   (
   TrqstHdl prmRqstHdl,
   TconnHdl prmConnHdl,
   Ts32     prmMaxBytes,
   Tbcb*    prmDataPtr
   );



/*
=================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
=================================================================

Name:
   typedef TrplyReceive

Description:
   The reply that corresponds to and returns results from the call to
   xptReceive().

Fields:
   mbrRplyId:
      Identifies to the receiver which kind of reply this is. Always
      equal to:
         ETrplyIdReceive
   mbrCCode:
      Identifies the conditions under which the reply was generated.
         ETcCodeSuccess:
            Data was received.
         ETcCodeFailure:
            Internal resources are unavailable to complete the
            request.  This may mean the connection has failed.  In
            that case TIAPI will generate a TrplyDisconnect for the
            connection.
         ETcCodeRequestCancelled:
```

```
            This request was canceled with a call to
            xptCancelRequest().
         ETcCodeInvalidConn:
            The specified connection handle is invalid.
   mbrRqstHdl:
      The request handle that was passed into TIAPI by OSP in the
      function call that corresponds to this reply.
   mbrConnHdl:
      Identifies which connection data was received over.
   mbrDataPtr:
      The same Tbcb* that was passed into TIAPI via the corresponding
      call to xptReceive().
      A pointer to a buffer containing the data that was received.
   mbrBytesReceived:
      The number of bytes received.

 ---------------------------------------------------------------
 */
 typedef  struct   TrplyReceive_
    {
    TrplyId  mbrRplyId;
    TcCode   mbrCCode;
    TrqstHdl mbrRqstHdl;
    TconnHdl mbrConnHdl;
    Tbcb*    mbrDataPtr;
    Ts32     mbrBytesReceived;
    }  TrplyReceive;


void xptCreateError
    (
    void *   prmErrorInfo,
    Tu32     prmErrorInfoLength,
    TconnHdl prmConnHdl
    );

typedef struct TrplyError_
    {
    TrplyId      mbrRplyId;
    TcCode       mbrCCode;
    TrqstHdl     mbrRqstHdl;
    TconnHdl     mbrConnHdl;
    Tbcb*        mbrErrorBuffer;
    }  TrplyError;


 /*
 ================================================
 Any TIAPI reply structure can be cast to this type.  When that is done
 the Trply structure can be examined to find out what type of reply it
 holds and the reply's status.  These 3 fields are common to all TIAPI
 reply structures and occur as the first 3 fields in all of them.
 ------------------------------------------------ */
 typedef  struct   TrplyHdr
    {
    TrplyId  mbrRplyId;
    TcCode   mbrCCode;
    TrqstHdl mbrRqstHdl;
    }  TrplyHdr;
 /*
 ================================================
 The Trply.mbrHeader.mbrRplyId field identifies which TIAPI reply
```

```
structure is contained in the Trply structure.  When that is known,
the corresponding mbr field can be referenced to examine the
information contained in the reply.
------------------------------------------------ */
typedef  union
   {
   TrplyHdr             mbrHeader;
   TrplyXlateOSPAddr    mbrRplyXlateOSPTI;
   TrplyXlateTIAddr     mbrRplyXlateTIOSP;
   TrplyNwkStat         mbrRplyNwkStat;
   TrplyActiveConnect   mbrRplyActiveConnect;
   TrplyPassiveConnect  mbrRplyPassiveConnect;
   TrplyDisconnect      mbrRplyDisconnect;
   TrplySend            mbrRplySend;
   TrplyReceive         mbrRplyReceive;
   TrplySendDatagram    mbrRplySendDatagram;
   TrplyReceiveDatagram mbrRplyReceiveDatagram;
   TrplyError           mbrError;
   }  Trply;



/*
====================================================================
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
====================================================================

Name:
   xptGetNextEvent()

Description:
   Pass the next TIAPI reply up to the OSP.

   OSP will call this function to retrieve replies from the TIAPI.
   TIAPI will store the replies that it generates on an internal queue
   and pass the next one in order up to the OSP when this function is
   called.

Parameters:
   prmRplyPtrPtr:
      A pointer to a pointer to a reply structure.  The reply
      structure pointer that is pointed to by prmRplyPtrPtr will be
      loaded with the address of the next reply from TIAPI.  Example:

         TIAPI owns a Trply named "rplyStruct" which it has filled in
         and is waiting to give to OSP.  OSP owns a pointer to
         a Trply named "P" that is not pointing to anything.  OSP
         passes a pointer to P to TIAPI with the call
         xptGetNextEvent(&P).  TIAPI loads P with a pointer to
         rplyStruct and returns.

         +-------------+   +-+   +----------+
         |prmRplyPtrPtr--->|P--->|rplyStruct|
         +-------------+   +-+   +----------+

            {
            Trply*  P;

            xptGetNextEvent(&P);

            // P now contains the address of (pointer to)
            // rplyStruct.
```

```
            }

   Returns:
      ETcCodeSuccess:
         If *prmRplyPtr contains a valid reply upon return.
      ETcCodeFailure:
         Otherwise.  OSP can assumed this to mean that there is no reply
         to retrieve at the time of the call.

   ------------------------------------------------------------------
   */
   TcCode   xptGetNextEvent
      (
      Trply**  prmRplyPtrPtr
      );



   /*
   ====================================================================
   [][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
   ====================================================================

   Name:
      xptNetworkEvent()

   Description:
      Process a network event.

      Network events are implementation defined.

      This function is called by the HP-JETSEND system when a network event
      has occured.  If xprNetworkEvent() causes any replies to be
      placed in the reply queue, it will call ospNetworkEvent(void) to
      allow OSP to retrieve the replies.

   Parameters:
      prmEventPtr:
         A pointer to a device defined event message.

   Returns:
      nothing.

   ------------------------------------------------------------------
   */
   void  xptNetworkEvent
      (
      void* prmEventPtr
      );



   #ifdef __cplusplus
      }
   #endif

#endif  /* #if !defined(_IGNORE_TIAPI_H) */
```