English

Fujitsu Software BS2000

XTI

X/Open Transport Interface

User Guide

Valid for:
XTI V6.0/V5.1

Edition December 2001

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to: bs2000.info@fujitsu.com

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Copyright © 2025 Fujitsu

All rights reserved.
Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

Preface

Overview of XTI

Address translation

Supplements to the function library

Supplements to the options

Supplements to event management

Read/write interface

XTI library trace

Reference section and index

Contents

1	**Preface**	**1**
1.1	Brief description of the product XTI	1
1.2	Target group	1
1.3	Summary of contents	1
1.4	README files	2
1.5	Changes since the last version of the manual	2
2	**Overview of XTI**	**3**
2.1	XTI components	3
2.1.1	Function library	3
2.1.2	Header files	5
2.1.3	xtil library trace program	6
2.1.4	Integration of XTI in the communication software	6
2.2	Runtime environment of XTI	7
2.3	Preparing XTI for operation	7
2.3.1	Installation	7
2.3.2	Compiling and linking	7
3	**Address translation**	**9**
3.1	Address management with TNS	9
3.1.1	TNS	9
3.1.2	TS directory	10
3.1.3	GLOBAL NAME	11
3.1.4	Properties	14
3.1.5	Transport services and LOCAL NAMES	16
3.2	Access functions t_getaddr(), t_getloc(), and t_getname()	19
3.2.1	t_getaddr() - get transport address	20
3.2.2	t_getloc() - get local name	23
3.2.3	t_getname() - get name	26
3.3	NETDIR access functions	29
3.3.1	netdir_getbyname() - map a GLOBAL NAME to a LOCAL NAME or to a TRANSPORT ADDRESS	30
3.3.2	netdir_getbyaddr() - map a TRANSPORT ADDRESS to a GLOBAL NAME	32
3.3.3	taddr2uaddr() - map a TRANSPORT ADDRESS to a universal address	34
3.3.4	uaddr2taddr() - map a universal address to a TRANSPORT ADDRESS	35
3.3.5	netdir_options() - interface to transport service options	36
3.3.6	netconfig - network configuration file	37

3.4 Address management with DIR.X 39
3.4.1 Introduction 39
3.4.2 The DIR.X Name Service 39
3.4.3 Requirements on communications applications 42
3.4.4 Parameterization of DIR.X 44
3.4.4.1 Facilities for the NSCONTROL variable 45
3.4.4.2 Facilities for the MAPRULES variable 49
3.4.5 Mapping to local address formats 49

4 Supplements to the function library 51
4.1 t_accept() - accept a connect request 54
4.2 t_alloc() - allocate a library structure 56
4.3 t_bind() - bind an address to a transport endpoint 57
4.4 t_connect() - establish a connection with another transport user 59
4.5 t_error() - produce error message 61
4.6 t_free() - free a library structure 62
4.7 t_getinfo() - get protocol-specific service information 63
4.8 t_listen() - listen for a connect request 64
4.9 t_look() - look at the current event on a transport endpoint . . . 66
4.10 t_open() - establish a transport endpoint 67
4.11 t_rcv() - receive data or expedited data sent over a connection . 73
4.12 t_rcvconnect() - receive the confirmation
from a connect request 74
4.13 t_rcvdis() - retrieve information from disconnect 76
4.14 t_rcvrel() - acknowledge receipt of an orderly release indication 80
4.15 t_rcvudata() - receive a data unit 81
4.16 t_rcvuderr() - receive a unit data error indication 82
4.17 t_snd() - send data or expedited data over a connection 83
4.18 t_snddis() - send user-initiated disconnect request 85
4.19 t_sndrel() - initiate an orderly release 86
4.20 t_sndudata() - send a data unit 87
4.21 t_sync() - synchronize transport library 88
4.22 t_sysconf() - get configurable XTI variables 89
4.23 t_unbind() - disable a transport endpoint 90

5 Supplements to the options 91
5.1 t_optmgmt() - manage options for a transport endpoint 91

6 Supplements to event management 95
6.1 poll() - multiplex input and output entities 96
6.2 select() - multiplex file descriptors 97

7 Read/write interface **99**
7.1 read() - read from a file 99
7.2 write() - write to file 100

8 XTI library trace **101**
8.1 XTITRACE - control the trace 102
8.2 xtil - edit the trace information 104

Glossary **107**

Abbreviations **113**

Tables **115**

References **117**

Index **119**

1 Preface

1.1 Brief description of the product XTI

XTI (X/Open Transport Interface) is the X/Open standard for an application program interface to transport systems. It is described in an X/Open Networking Services (XNS) [5] that is part of the UNIX specification. This product implements XTI in your Reliant UNIX / Solaris system.

1.2 Target group

This manual is intended for programmers of communications applications. Communications applications are C user programs that exchange messages with other communications applications in the network.

The programmer is expected to have a good knowledge of the C programming language. A knowledge of the principles and methods of teleprocessing would also be helpful, in particular of the OSI Reference Model as standardized in ISO 7498.

1.3 Summary of contents

This manual complements the X/Open Networking Services (XNS) [5] and should be used in conjunction with it. It deals with those aspects which are not defined in the X/Open standard and are thus implementation-defined. These aspects are, for example, the names of the transport providers, addresses, supported options, and trace facilities.

When creating XTI applications, therefore, you must refer to the “X/Open Networking Services (XNS)” [5].

1.4 README files

For functional modifications and supplements to the current product version not included in this manual, you may have to refer to the product-specific README files. These are located on your Reliant UNIX system under the directory */opt/readme/XTI* and on your Solaris system under the directory *opt/SMAW/documents/CMX*, provided that your system administrator installed the README supplied with the product. The README files can be read using an editor, or printed on a standard printer.

1.5 Changes since the last version of the manual

The system Solaris has been added. The expression UNIX in the manual corresponds to Reliant UNIX and Solaris.

2 Overview of XTI

The product XTI is the implementation of the X/Open Transport Interface (see the “X/Open Networking Services (XNS)” [5]), and provides communications applications with a program interface for uniform access to all transport services. Communications applications are transport system users as defined by X/Open. The interface has been implemented as a set of C functions which are contained in a function library. The name of the function library is specified in the Release Notice.

This chapter provides an overview of how XTI is integrated in the communication software of your end system. It describes the environment required for developing and running XTI applications.

2.1 XTI components

The software product XTI V5.1 comprises a function library, the header files *<xti.h>, <xti_inet.h>* and *<xti_osi.h>* and the library trace program *xtil*.

2.1.1 Function library

This library contains the general XTI calls and the functions for accessing the Name Service. General XTI calls are:

XTI call	Meaning
t_accept	accept a connect request
t_alloc	allocate a library structure
t_bind	bind an address to a transport endpoint
t_close	close a transport endpoint
t_connect	establish a connection with another transport user
t_error	produce error message
t_free	free a library structure
t_getinfo	get protocol-specific service information
t_getprotaddr	get protocol address

Table 1: General XTI calls

XTI call	Meaning
t_getstate	get current state
t_listen	listen for a connect request
t_look	look at the current event on a transport endpoint
t_open	establish a transport endpoint
t_optmgmt	manage options for a transport endpoint
t_rcv	receive data or expedited data sent over a connection
t_rcvconnect	receive the confirmation from a connect request
t_rcvdis	retrieve information from disconnect
t_rcvrel	acknowledge receipt of an orderly release indication
t_rcvreldata	receive an orderly release indication and any user data sent with the release
t_rcvudata	receive a data unit
t_rcvuderr	receive a unit data error indication
t_rcvvudata	receive a data unit in non-contiguous buffers
t_rcvv	receive normal or expedited data in non-contiguous buffers
t_snd	send data or expedited data over a connection
t_snddis	send user-initiated disconnect request
t_sndrel	initiate an orderly release
t_sndreldata	initiate an orderly release sending user data with the release
t_sndudata	send a data unit
t_sndvudata	send a data unit from non-contiguous buffers
t_sndv	send normal or expedited data from non-contiguous buffers
t_strerror	supply error message text
t_sync	synchronize transport library
t_sysconf	determine the current value of configurable and implementation-dependent XTI limits.
t_unbind	disable a transport endpoint

Table 1: General XTI calls

The general XTI calls are described in the “X/Open Networking Services (XNS)” [5].

Functions for accessing the Name Service:

Function	**Meaning**
t_getaddr	get transport address
t_getloc	get local name
t_getname	get global name
netdir_getbyname	map a GLOBAL NAME to a LOCAL NAME or to a TRANSPORT ADDRESS
netdir_getbyaddr	map a TRANSPORT ADDRESS to a GLOBAL NAME
taddr2uaddr	map a TRANSPORT ADDRESS to a universal address
uaddr2taddr	map a universal address to a TRANSPORT ADDRESS
netdir_options	interface to transport service options
netconfig	network configuration file

Table 2: Functions for accessing the Name Service

The functions for accessing the Name Service are described in section “Access functions t_getaddr(), t_getloc(), and t_getname()” on page 19.

2.1.2 Header files

The header files contain the structures and constants used by XTI. The following header files are available:

Header file	**Content**
<xti.h>	general definitions
<xti_inet.h>	IP-specific definitions
<xti_osi.h>	ISO-specific definitions

Table 3: Header files and applications

For implications of the header files *<xti.h>*, *<xti_inet.h>* and *<xti_osi.h>* refer to section “Compiling and linking” on page 7 and also to the “X/Open Networking Services (XNS)” [5].

2.1.3 xtil library trace program

This processes information from the XTI library trace. See also chapter “XTI library trace” on page 101.

2.1.4 Integration of XTI in the communication software

The product is embedded in the CMX infrastructure (see figure 1). To access an ISO, ISO-over-TCP (RFC1006), or NEA transport service, the XTI function library calls the CMX automaton in the operating system kernel. The CMX automaton coordinates the communication activities in the system and relays the messages to the correct transport service provider.

The situation is different for TCP/IP and UDP/IP. The XTI function library directly accesses these protocols via a STREAMS system interface.

A schematic overview is given in figure 1.

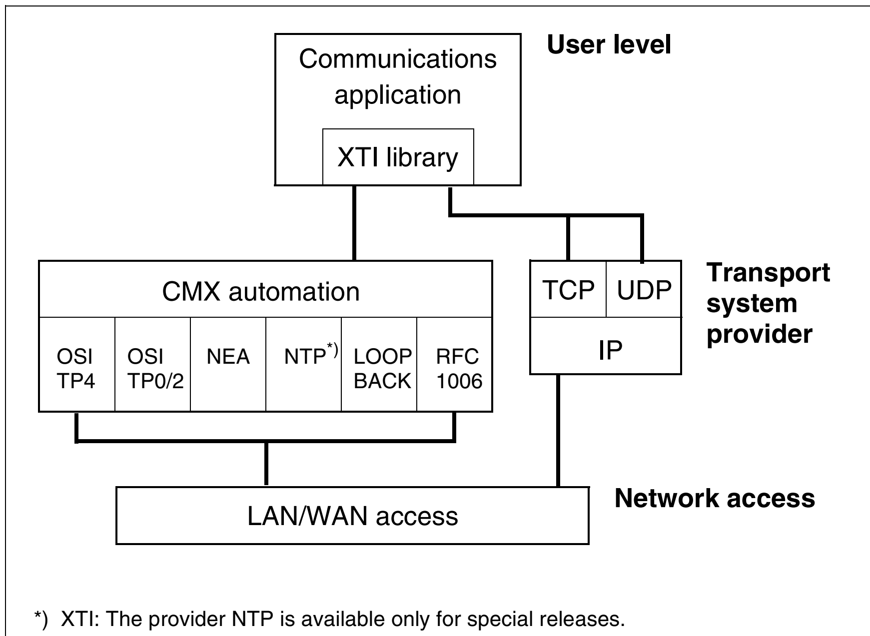


Figure 1: Integration of XTI in the communication software

2.2 Runtime environment of XTI

In order to **develop** communications applications with XTI, all you need to do is to install XTI V5.1, i.e. you can write, compile, and link source programs using the XTI function library.

The environment required by an XTI application at **runtime** depends on the transport systems used for communication.

This XTI product provides access to the OSI (including RFC1006 over TCP/IP), NEA, TCP/IP (native), UDP/IP, and loop-back transport services in your system. These services are not part of this product; they are either supplied with the UNIX operating system (as TCP/IP or UDP) or can be ordered separately. CMX is needed if your application programs are meant to work with transport systems different from UDP/IP and TCP/IP (native).

2.3 Preparing XTI for operation

2.3.1 Installation

The installation of XTI depends on the respective operating system version and is described in the Release Notice. The installation of XTI is sufficient for developing XTI applications and to run XTI applications across TCP/IP and UDP/IP. The product CMX must be installed if communication across RFC1006-over-TCP is required. To use other transport systems, CMX and the respective product of the CCP product family must be installed.

In most cases, the communication addresses of the applications are administered by the Transport Name Service (TNS), a component on CMX. Details concerning the administration of communication addresses can be found in the manual “CMX, Operation and Administration” [1].

2.3.2 Compiling and linking

By including the header file *<xti.h>* into an application source program, all structures and constants relevant for XTI usage are made available. All you need to do is to enter the statement inside the source program *xti.prog.c*:

inside *xtiprog.c* enter: #include *<xti.h>*

By default *<xti.h>* encompasses generic XTI symbols as well as definitions that are specific for OSI transport protocols respectively Internet protocols. If a communication application does not require (all of) the protocol dependent symbols, XTI 5.1 offers the following means to reduce the overhead within *<xti.h>*:

– By specifying option “-D_XOPEN_SOURCE=500” when compiling the application source program the header file *<xti.h>* will be reduced to contain generic XTI symbols only, thus omitting all protocol-dependent definitions. This should suffice for most applications except those that use protocol-specific features such as option handling. All you need to do is to enter the following two statements inside the source program *xti.prog.c* respectively when compiling the source program:

inside *xtiprog.c* enter:	#include *<xti.h>*
when compiling *xtiprog.c* enter:	cc *xtiprog.c* .... -D_XOPEN_SOURCE=500 ...

– If a communications application is specifically designed for a certain protocol- group (OSI or IP), use option “-D_XOPEN_SOURCE=500” at compilation time and include one of the protocol-specific header files *<xti_inet.h>* respectively *<xti_osi.h>* to your application program.

inside *xtiprog.c* enter:	#include *<xti.h>* and
	#include *<xti_osi.h>* respectively
	#include *<xti_inet.h>*
when compiling *xtiprog.c*	
enter	cc *xtiprog.c* .... -D_XOPEN_SOURCE=500 ...

In a system with the X/Open UNIX brand, the XTI functions are recognized by the C compiler if it is called with the operand *-lxnet*. The operand may be different in non-branded systems. The release notice describes exactly which operand should be used depending on the UNIX version.

3 Address translation

3.1 Address management with TNS

3.1.1 TNS

The TNS is not necessary (though useful) for communication via TCP/IP and UDP/IP.

The Transport Name Service TNS provided by CMX can be used to manage the names and addresses of communications applications. The TNS reads the address information from a directory known as the TS directory (Transport System directory). The address information for each communications application is stored here under its symbolic name, i.e. the GLOBAL NAME of the communications application. The TS directory must contain information on all communications applications residing in the local system and on potential communication partners in remote systems.

The User Interfaces CMXGUI, CMXCUI and CMX command line interface support the system administrator when creating and maintaining this directory. They are described in the manual “CMX, Operation and Administration” [1]. This section describes the TS directory (directory of names and addresses), the names and properties of communications applications managed by TNS, and the function calls used for obtaining names and addresses from TNS.

The X/Open standard XTI does not define how the appropriate addresses are to be supplied to the structures involved in linking up with an access point to the transport services and in establishing a transport connection. The communications application must either pass the transport address of the remote communications application to XTI or assign (“bind”) a protocol address to a transport endpoint. For communication via transport service providers that are accessed via CMX, these addresses are managed in a register of names and addresses, the TS directory. In case of TCP/IP and UDP/IP, these addresses can either be managed in the TS directory itself or generated by the communications application (see “Features of TCP/IP and UDP/IP” on page 17). The TNS service (Transport Name Service in UNIX) manages the addresses in the TS directory. The TNS is described in the manual “CMX, Operation and Administration” [1]).

Each communications application, whether local or remote, is assigned a logical name. These names permit the communications applications to be uniquely identified throughout the network, i.e. different communications applications have different names. These names are called the GLOBAL NAMES of the communications applications. The GLOBAL NAME identifies the communications application concerned in a way suitable for the user. The GLOBAL NAMES of all communications applications in the local system and all communications applications in remote systems with which the local communications applications wish to communicate are registered in the TS directory. The structure of the GLOBAL NAMES is described in a later section.

In the TS directory, the GLOBAL NAMES are assigned the properties of the corresponding communications applications. A local communications application has the property LOCAL NAME. The LOCAL NAME must be passed to XTI together with the *t_bind()* call. A remote communications application has the property TRANSPORT ADDRESS. The TRANSPORT ADDRESS is passed to XTI during the connection establishment phase. LOCAL NAME and TRANSPORT ADDRESS are machine-oriented data, their format and contents depend on the underlying communication network and its current configuration. In order to be independent of these, communications applications use the GLOBAL NAMES only. They issue special function calls to query the LOCAL NAMES and TRANSPORT ADDRESSES associated with the GLOBAL NAMES and pass them on to XTI unchecked. A communications application can likewise obtain the GLOBAL NAME associated with a TRANSPORT ADDRESS.

3.1.2 TS directory

The TS directory consists of entries each of which contains information about a communications application in the form of properties. The entries are identified by the GLOBAL NAME of the communications application and are arranged in the form of a naming tree.

The TNS supports up to 9 different TS directories with the identifications 1-9. The TS directories are stored in the file system as directories:

DIR<id>
    <id> = identification = 1,...,9

See manual “CMX, Operation and Administration” [1] for the information where these directories are stored in your system’s file system.

Using one of the User Interfaces, the system administrator defines one of the TS directories as the standard TS directory.

It is possible to list such a directory with the usual UNIX tools, but the files contained there are made up primarily of non-printable information. However, the TS directories can be formatted into printable text (see manual “CMX, Operation and Administration” [1]). The CMX User Interfaces can also be used by the system administrator to create and update a TS directory.

i A TS directory cannot be deleted or transported to another computer using the normal tools provided by the system. The functions of the User Interfaces (see manual “CMX, Operation and Administration” [1]) must be used to do this. Only these interfaces ensure that the caches are synchronized correctly with the files on secondary storage.

3.1.3 GLOBAL NAME

The GLOBAL NAME of a communications application is a hierarchically structured name. It consists of any subset of up to 5 name parts, name part[1] (NP1) through name part[5] (NP5). Of these, name part[1] is the highest in the hierarchy, name part[5] the lowest. Not all levels of the hierarchy need be present in a GLOBAL NAME. Apart from the hierarchical order, TNS makes no further specifications regarding the meanings of the name parts within a GLOBAL NAME. The hierarchical structure permits the GLOBAL NAMES to be arranged in a naming tree (the “global naming tree”). The figure “Example of a naming tree of GLOBAL NAMES” on page 12 shows an example of a naming tree.

The tree consists of three elements:

– the root (ROOT),

– the nodes, also called NonLeafEntities,

– the leaves, also called LeafEntities.

The tree grows top down.

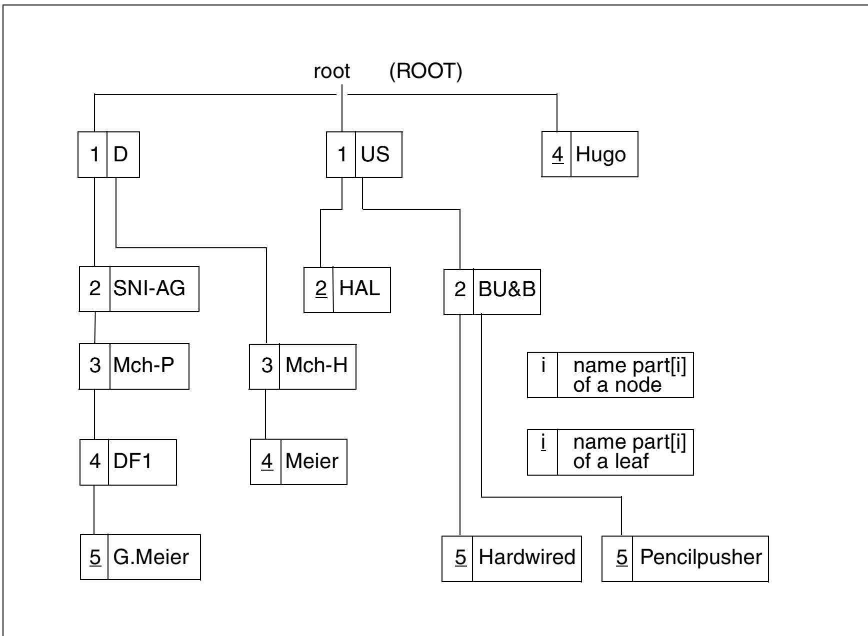


Figure 2: Example of a naming tree of GLOBAL NAMES

This results in a GLOBAL NAME having the following features:

- A GLOBAL NAME corresponds to a path in a naming tree from the root to a leaf or node, with the name parts corresponding to the path components.
- Following the hierarchy, an additional name part can be attached to the root or to a node, simultaneously determining whether the resulting path is to lead to a node or a leaf. A name part thus leads from the root or from a node to another node or to a leaf.
- All name parts can be path components leading to a leaf. Except for name part[5], all name parts can be path components leading to a node.
- Properties can be assigned to a leaf only.

In the function calls described in section “Access functions t_getaddr(), t_getloc(), and t_getname()” on page 19, the GLOBAL NAME must be specified as a null-terminated character string of the following format

“NP5.NP4.NP3.NP2.NP1”

The NPi (i=1,2,3,4,5) are the name parts of the GLOBAL NAME, with NP5 being name part[5], the name part of the lowest hierarchical level. NP1 is name part[1], i.e. the name part of the highest hierarchical level. The remaining name parts must be specified in ascending hierarchical order, starting from left to right.

If any of the name parts of a GLOBAL NAME is not used (e.g. NP4) but followed by another name part of a higher hierarchical level (e.g. NP3), the separator (.) belonging to the omitted name part must be specified. A sequence of separators at the end of a GLOBAL NAME can be omitted.

The GLOBAL NAME would in this case be specified as: “NP5..NP3”.

In figure “Example of a naming tree of GLOBAL NAMES” on page 12, for instance, the leaf “Hugo” given as name part[4] appears directly under the root. This GLOBAL NAME is passed as “.Hugo”.

If the character which serves as the separator, i.e. the period (.), is used within a name part (as in G.Meier) it must be specified as \. (backslash period).

Examples:

1. GLOBAL NAME: name part[1] = D
   name part[2] = SNI-AG
   name part[3] = MCH-P
   name part[4] = DF1
   name part[5] = G.MEIER

   Specify as: G\.MEIER.DF1.MCH-P.SNI-AG.D”

1. GLOBAL NAME: name part[1]= US
   name part[2]= BU&B
   name part[5]= PENCILPUSHER

   Specify as: PENCILPUSHER...BU&B.US”

The GLOBAL NAMES are assigned by the administration.

3.1.4 Properties

The GLOBAL NAMES representing the communications applications in the TS directory are assigned the properties shown in figure 3. The TNS also recognizes user-specific properties, which can be entered, if required, in the TS directory using a CMX User Interface. However, XTI applications cannot read these properties from the TS directory.

This section only describes the properties relevant for the developer of XTI applications. A detailed description of all properties is given in the manual "CMX, Operation and Administration" [1]).

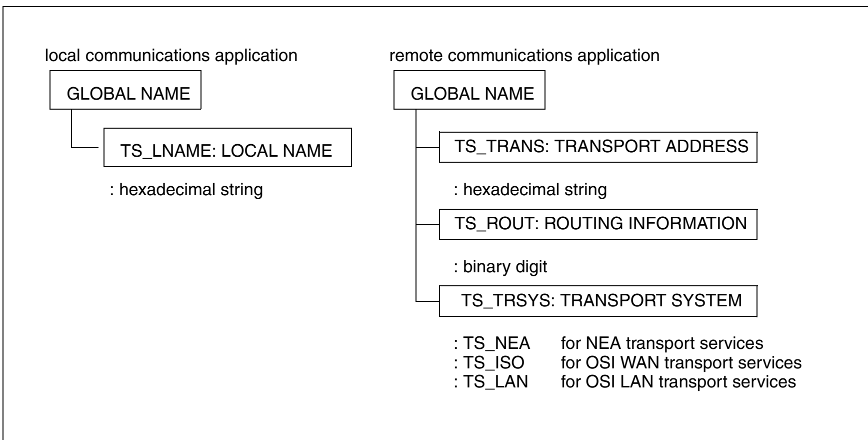


Figure 3: Properties of communications applications

Properties of a local communications application

A local communications application is entered in the TS directory with the property LOCAL NAME.

The LOCAL NAME property of a communications application is required in order to bind this communications application to a transport provider. This property must be passed to XTI when *t_bind()* is called. It comprises one or more T-selectors, which are the addresses of a communications application in the local end system for the various transport service providers. The LOCAL NAME is a hexadecimal string of non-printable characters. For a more detailed description see below.

The LOCAL NAME of a communications application can be established from the GLOBAL NAME by calling *t_getloc()*.

Properties of a remote communications application

For a **remote** communications application, i.e. an application residing in another system, the properties TRANSPORT ADDRESS, ROUTING INFORMATION and TRANSPORT SYSTEM are recorded in the TS directory.

The value of the TRANSPORT ADDRESS property is the communication partner's address which XTI expects at connection setup. It is read from the TS directory by calling *t_getaddr()*. The call *t_getname()* serves to convert the TRANSPORT ADDRESS into the communication partner's GLOBAL NAME.

The TRANSPORT ADDRESS is a hexadecimal string of non-printable characters. For a more detailed description see below.

The properties LOCAL NAME and TRANSPORT ADDRESS

A transport service access point (abbreviated TSAP) is uniquely assigned to a communications application when *t_bind()* is called. The TSAP is identified by the LOCAL NAME of the communications application. The TSAP can be used by the communications application to access the transport service provider for communication purposes. The T-selectors contained in the LOCAL NAME determine which transport service provider, i.e. which network interface, can be accessed by the communications application. A T-selector can be valid for more than one transport service provider.

The communications application can be addressed from the network using the T-selector. The T-selector is part of the TRANSPORT ADDRESS of the respective network. The communications application can be uniquely addressed throughout the network using the TRANSPORT ADDRESS.

In accordance with OSI regulations, the TRANSPORT ADDRESS comprises the network address of the end system containing the communications application and the T-selector unique in this end system:

TRANSPORT ADDRESS = network address of end system + T-selector

The diagram below clarifies the relationship between LOCAL NAME, TRANSPORT ADDRESS and TSAP.

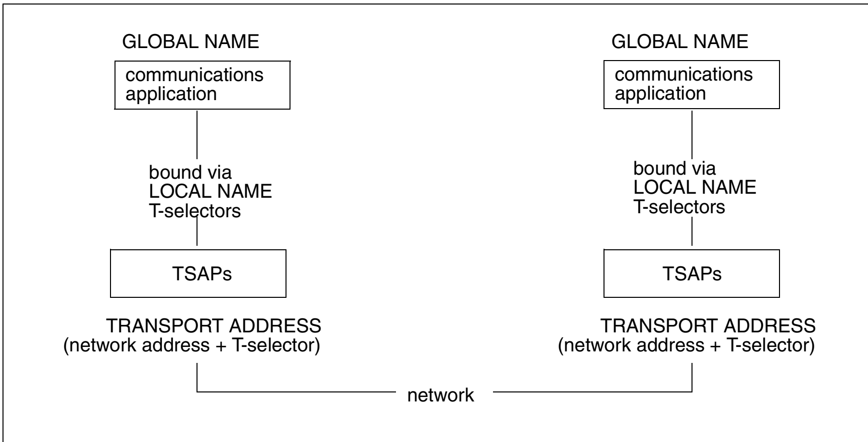


Figure 4: TRANSPORT ADDRESS and LOCAL NAME

The sections ""Transport services and LOCAL NAMES" on page 16" and ""t_open() - establish a transport endpoint" on page 67" describe the required structure of a LOCAL NAME, i.e. the T-selectors it must contain if it is to match the transport service specified in the *t_open()* call. The section on "Transport services and LOCAL NAMES" also indicates points to be observed when assigning a LOCAL NAME.

The required composition of the T-selectors for the individual transport providers is described in the manual "CMX, Operation and Administration" [1]).

3.1.5 Transport services and LOCAL NAMES

As described in section "Properties" on page 14, the LOCAL NAME of a communications application consists of one or more T-selectors, each of which refers to a specific group of transport providers (CCPs). In the section "t_open() - establish a transport endpoint", table "Association of transport services with CCP profiles" on page 69 shows which T-selectors are of significance for the individual transport services. Similar rules apply to the TRANSPORT ADDRESSES. These contain a type field whose value must match the transport service.

Communications applications for different transport services using the same LOCAL NAME

ISO and NEA transport addresses and TCP/IP or UDP/IP transport addresses cannot be bound to a transport endpoint at the same time. The following convention therefore applies:

If a LOCAL NAME includes entries for the TCP/IP or UDP/IP type as well as for the other types, *t_getloc()* only returns the entries for the TCP/IP and UDP/IP types.

When defining the LOCAL NAME, it is recommended not to include both entries for the TCP/IP and UDP/IP types and entries for other types.

In accordance with XTI rules, several communications applications using the same transport service may be bound to the same LOCAL NAME. Since a LOCAL NAME may have more than one T-selector (each one referring to a different transport service), even applications using different transport services could bind to the same LOCAL NAME. This should be avoided, however, since interference effects could result. If both applications tried to bind with *qlen > 0*, e.g., then one *t_bind()* call would fail with [TADDRBUSY], although the transport endpoints belong to different transport services.

Features of TCP/IP and UDP/IP

The LOCAL NAME and TRANSPORT ADDRESS of TCP/IP and UDP/IP have the structure *sockaddr_in* (defined in <netinet/in.h>).

```
struct in_addr
{
       u_long      s_addr;
};

struct sockaddr_in
{
       short             sin_family;   /* protocol family */
       u_short           sin_port;        /* port number */
       struct in_addr    sin_addr;        /* host address */
       char              sin_zero[8];
};
```

The LOCAL NAME and TRANSPORT ADDRESS can be managed with the usual mechanisms (t_getaddr(), t_getloc(), t_getname()) with TNS.

However, the communications application can generate the LOCAL NAME and TRANSPORT ADDRESS itself, whereby the same rules apply as for socket interface addresses. It must be ensured here that the correct byte order is retained.

The address fields must be allocated as follows to create a LOCAL NAME:

sin_family:
AF_INET (defined in <sys/socket.h>)

sin_port:
Port number. Port numbers between 0 and 1023 are reserved for superusers. If 0 is specified, the system assigns any free port number greater than 1023.

sin_addr.s_addr:
Local host address. The specification INADDR_ANY means “any valid address”. INADDR_ANY is particularly useful if the system has several network accesses.

sin_zero[8]:
Assign zeros.

The following must be noted when creating a transport address:

The fields *sin_port* and *sin_addr.s_addr* designate the partner application’s port number and host address respectively. The specification INADDR_ANY is not permitted. The remaining fields must be filled as for LOCAL NAME.

3.2 Access functions t_getaddr(), t_getloc(), and t_getname()

TNS, the Transport Name Service in UNIX, can be used to obtain the LOCAL NAME or the TRANSPORT ADDRESS associated with a particular GLOBAL NAME, or to ascertain the GLOBAL NAME of a communication partner starting from the partner's TRANSPORT ADDRESS. XTI V5.1 offers functions which make these TNS queries required for XTI possible.

The following calls will be described in this section:

t_getaddr()
supplies the TRANSPORT ADDRESS of the communications application whose GLOBAL NAME is specified. The TRANSPORT ADDRESS is required as a parameter by a number of XTI calls.

t_getloc()
supplies the LOCAL NAME in the local end system of a communications application whose GLOBAL NAME is specified. The LOCAL NAME is required as a parameter by a number of XTI calls.

t_getname()
supplies the GLOBAL NAME of a communications application whose TRANSPORT ADDRESS is specified.

Metasyntax of the three calls

The following symbols are used in the description of the function calls:

-> indicates parameters whose value must be supplied by the caller.

<- indicates parameters whose value is supplied by the function.

<> indicates parameters which require the caller to submit a value that is subsequently modified by the function.

3.2.1 t_getaddr() - get transport address

Synopsis

```
#include <xti.h>
int t_getaddr(globname, addr, opt)
char *globname;
struct netbuf *addr;
struct netbuf *opt;
```

Description

t_getaddr() obtains the TRANSPORT ADDRESS of a remote communications application from the TS directory. The GLOBAL NAME of the communications application must be specified in the *globname* parameter.

The result returned by *t_getaddr()*, i.e. the TRANSPORT ADDRESS of the communications application, is entered in the *addr* structure in the format required by the *t_connect()* call, for example. The TRANSPORT ADDRESS supplied is passed directly to XTI with *t_connect()*.

The individual parameters have the following meaning:

-> globname

The GLOBAL NAME of the communications application whose TRANSPORT ADDRESS is to be obtained must be specified in this parameter. The GLOBAL NAME must be specified as a null-terminated character string of the following format: “NP5.NP4.NP3.NP2.NP1”

NPi (i=1,2,3,4,5) are the name parts of the GLOBAL NAME. The name parts must be specified in ascending hierarchical order, starting from left to right (see section “GLOBAL NAME” on page 11).

If any of the name parts of the GLOBAL NAME is not used the separator (.) belonging to the omitted name part must be specified. A sequence of separators at the end of the value of *globname* can be omitted. At least one name part NPi must be specified.

If the character which serves as the separator, i.e. the period (.), is used within a name part it must be specified as \. (backslash period).

<> addr

points to an element of type *struct netbuf*. The maximum length of the buffer specified with *buf* must be entered in the *maxlen* member of this structure when issuing the call. If the maximum length is too small, an error message is returned. The maximum length of the TRANSPORT ADDRESS is supplied in *info->addr* by the calls *t_open()* and *t_getinfo()*. The actual length of the TRANSPORT ADDRESS stored in *buf* is returned in *len*.

-> opt

must be the null pointer (reserved for future extensions).

t_getaddr() also provides access to the X.500 Directory Service via DIR.X ≥V3.0. Instead of a GLOBAL NAME, you can also specify an X.500 Distinguished Name in *globname*. Further details and restrictions are given in section “Address management with DIR.X” on page 39.

Return value

t_getaddr() returns the value 0 if the search was successful and the value -1 in the event of an error. In the latter case, an error indication is entered in *t_errno*.

Error

In the event of an error, one of the following values is entered in *t_errno*:

[TBADNAME]

The GLOBAL NAME specified in *globname* has the wrong format or contains illegal values or cannot be found or internal error when accessing the TS directory.

[TBUFOVFLW]

The length specified in *maxlen* is insufficient for the TRANSPORT ADDRESS. The maximum length permissible for the TRANSPORT ADDRESS can be queried with *t_getinfo()*.

[TNOADDR]

No TRANSPORT ADDRESS associated with the GLOBAL NAME specified with *globname* could be found.

[TSYSERR]

A system error occurred when accessing the TS directory. The value of the error is entered in the variable *errno*.

If *t_errno* contains the value [TBADNAME] and you suspect that an internal error occurred when accessing the TS directory, you should activate the XTI library trace for a more detailed diagnosis. This is described in chapter “XTI library trace” on page 101. The error type, error class, and error value will be logged in the trace file produced by *xtil*.
If CMX is installed on your system, you can use the program *cmxdec* at command level to request diagnostic information in plain text on the error type, error class and error value.

See also

t_connect(), t_getinfo(), t_open(); section “GLOBAL NAME” on page 11; manual “CMX, Programming Applications” [2].

3.2.2 t_getloc() - get local name

Synopsis

```
#include <xti.h>
int t_getloc(globname, addr, opt)
char *globname;
struct netbuf *addr;
struct netbuf *opt;
```

Description

t_getloc() obtains the LOCAL NAME of a communications application from the TS directory. The GLOBAL NAME of the communications application must be specified in the *globname* parameter.

The result returned by *t_getloc()*, i.e. the LOCAL NAME of the communications application, is entered in the *addr* structure in the format required by the t_bind() call, for example. The LOCAL NAME supplied is passed directly to XTI with *t_bind()*.

The individual parameters have the following meaning:

-> globname

The GLOBAL NAME of the communications application whose LOCAL NAME is to be obtained must be specified in this parameter. The GLOBAL NAME must be specified as a null-terminated character string of the following format: “NP5.NP4.NP3.NP2.NP1”

NPi (i=1,2,3,4,5) are the name parts of the GLOBAL NAME. The name parts must be specified in ascending hierarchical order, starting from left to right (see section “GLOBAL NAME” on page 11).

If any of the name parts of the GLOBAL NAME is not used the separator (.) belonging to the omitted name part must be specified. A sequence of separators at the end of the value of *globname* can be omitted. At least one name part NPi must be specified.

If the character which serves as the separator, i.e. the period(.), is used within a name part it must be specified as \. (backslash period).

<> addr

points to an element of type *struct netbuf*. The maximum length of the buffer specified with *buf* must be entered in the *maxlen* member of this structure when issuing the call. If the maximum length is too small, an error message is returned. The maximum length of the LOCAL NAME is supplied in *info->addr* by the calls *t_open()* and *t_getinfo()*. The actual length of the LOCAL NAME stored in *buf* is returned in *len*.

-> opt

must be the null pointer (reserved for future extensions).

t_getloc() also provides access to the X.500 Directory Service via DIR.X ≥V3.0. Instead of a GLOBAL NAME, you can also specify an X.500 Distinguished Name in *globname*. Further details and restrictions are given in section “Address management with DIR.X” on page 39.

Return value

t_getloc() returns the value 0 if the search was successful and the value -1 in the event of an error. In the latter case, an error indication is entered in *t_errno*.

Error

In the event of an error, one of the following values is entered in *t_errno:*

[TBADNAME]

The GLOBAL NAME specified in *globname* has the wrong format or contains illegal values or cannot be found; or internal error when accessing the TS directory.

[TBUFOVFLW]

The length specified in *maxlen* is insufficient for the LOCAL NAME. The maximum length permissible for the LOCAL NAME can be queried with *t_getinfo()*.

[TNOADDR]

No LOCAL NAME associated with the GLOBAL NAME specified with *globname* could be found.

[TSYSERR]

A system error occurred when accessing the TS directory. The value of the error is entered in the variable *errno*.

If *t_errno* contains the value [TBADNAME] and you suspect that an internal error occurred when accessing the TS directory, you should activate the XTI library trace for a more detailed diagnosis. This is described in chapter “XTI library trace” on page 101. The error type, error class, and error value will be logged in the trace file produced by *xtil*.

If CMX ≥V4.0 is installed on your system, you can use the program *cmxdec* at command level to request diagnostic information in plain text on the error type, error class and error value.

See also

t_bind(), t_getinfo(), t_open(); section “GLOBAL NAME” on page 11; appendix in manual “CMX, Programming Applications” [2].

3.2.3 t_getname() - get name

Synopsis

```
#include <xti.h>
char *t_getname(addr, opt)
struct netbuf *addr;
struct netbuf *opt;
```

Description

t_getname() obtains the GLOBAL NAME of a communications application from its TRANSPORT ADDRESS. *t_getname()* writes the GLOBAL NAME to a static area and supplies the running communications application with the pointer to that area. The static area is overwritten with each call. It must be copied if it is to be saved.

Note that the function must be declared since it is not of type “integer”.

The GLOBAL NAME is returned as a null-terminated character string of the following format:

“Np5.Np4.Np3.Np2.Np1”

NPi (i=1,2,3,4,5) are the name parts of the GLOBAL NAME. The name parts are specified in ascending order, starting from left to right (see section “GLOBAL NAME” on page 11).

If any of the name parts of the GLOBAL NAME is not used the separator (.) belonging to the omitted name part is specified unless the omitted name part is the final name part. A sequence of separators at the end of the GLOBAL NAME is omitted.

If the character which serves as the separator, i.e. the period (.), is used within a name part it is represented as \. (backslash period).

The individual parameters have the following meaning:

-> addr

points to an element of type *struct netbuf* with the following contents:

len contains the length of the TRANSPORT ADDRESS submitted

buf contains the pointer to the buffer containing the TRANSPORT ADDRESS

maxlen

has no meaning for this function.

-> opt

must be the null pointer (reserved for future extensions).

t_getname() also provides a name search facility in the X.500 Directory Service via DIR.X ≥V3.0. An X.500 Distinguished Name is then returned instead of a GLOBAL NAME. For further details and restrictions, see section “Address management with DIR.X” on page 39.

Return value

t_getname() returns a pointer to the GLOBAL NAME if the search was successful and the null pointer in the event of an error. In the latter case, an error indication is entered in *t_errno*.

Error

In the event of an error, one of the following values is entered in *t_errno*:

[TBADADDR]

The TRANSPORT ADDRESS specified in *addr* has the wrong format or contains illegal values.

[TNOADDR]

The TRANSPORT ADDRESS referred to in *addr* cannot be found

or

internal error when accessing the TS directory.

[TSYSERR]

A system error occurred when accessing the TS directory. The value of the error is entered in the variable *errno*.

If *t_errno* contains the value [TNOADDR] and you suspect that an internal error occurred when accessing the TS directory, you should activate the XTI library trace for a more detailed diagnosis. This is described in chapter “XTI library trace” on page 101. The error type, error class and error value will be logged in the trace file produced by *xtil*.

If CMX≥V4.0 is installed on your system, you can use the program *cmxdec* at command level to request diagnostic information in plain text on the error type, error class and error value.

See also

t_getaddr(), t_listen(), t_connect(), t_rcvconnect(); section “GLOBAL NAME” on page 11; appendix in manual “CMX, Programming Applications” [2].

3.3 NETDIR access functions

XTI applications can also use the NETDIR access interface instead of the functions described in section “Access functions t_getaddr(), t_getloc(), and t_getname()” on page 19. The NETDIR interface is described in detail in chapter 4 “IP Address Resolution Interfaces” of the “X/Open Networking Services (XNS)” [5]. Its functions originated in System V Release 4, and have now been incorporated in the XTI library.

The NETDIR interface offers uniform, flexible access to various Name Service directories.

This section describes how to use NETDIR to access the TNS. For a general description of the interface, refer to chapter 4 “IP Address Resolution Interfaces” of the “X/Open Networking Services (XNS)” [5].

For further description see manual “Reliant UNIX 5.43, Network Programming Interfaces” [3] and “Solaris 7 Reference Manual collection” [4].

3.3.1 netdir_getbyname() - map a GLOBAL NAME to a LOCAL NAME or to a TRANSPORT ADDRESS

This function is used to map a GLOBAL NAME to a LOCAL NAME or to a TRANSPORT ADDRESS.

Synopsis

```
#include <netdir.h>
int netdir_getbyname(config, service, addrs)
struct netconfig *config;
struct nd_hostserv *service;
struct nd_addrlist **addrs;
```

Implementation-specific supplements

GLOBAL NAMES managed with the TNS have the following format:

Np5.Np4.Np3.Np2.Np1

Npi are the name parts of the GLOBAL NAME in ascending hierarchical order, starting from left to right. If a name part is not used, but is followed by at least one name part that is used, the separator '.' must be inserted between two name parts (a sequence of '.' characters at the end of the GLOBAL NAME can be omitted). If the '.' is used within a name part, it must be specified as \. (backslash period).

The following rules apply for GLOBAL NAMES of communications applications running in the **local** end system:

– Entering the GLOBAL NAME:

  Np1, Np2 and Np3 are empty.

  Np4 identifies the local end system; this is preset to the name of the system as supplied by *uname -n*.

  Np5 is a (mandatory) arbitrary value and identifies the communications application within the local end system.

– Calling *netdir_getbyname()*:

  service->h_host points to a character string that must contain the constant HOST_SELF.

  service->h_serv points to a character string that contains Np5.

  If the *netdir_getbyname()* function was successful, *(*addrs)->n_cnt*

  has the value 1 and *(*addrs)->n_addrs* points to a *struct netbuf* containing the LOCAL NAME of the communications application.

The following rules apply for GLOBAL NAMES of communications applications running in a **remote** end system:

– Entering the GLOBAL NAME:

  Np1, Np2, Np3 and Np4 are arbitrary values (at least one name part must be specified) and identify the remote end system.

  Np5 is a (mandatory) arbitrary value and identifies the communications application within the remote end system.

– Calling *netdir_getbyname()*:

  service->h_host points to a character string that contains Np4.Np3.Np2.Np1.

  service->h_serv points to a character string that contains Np5.

  If the *netdir_getbyname()* function was successful, *(*addrs)->n_cnt* has the value 1 and *(*addrs)->n_addrs* points to a *struct netbuf* containing the TRANSPORT ADDRESS of the communications application.

If the function is used to access an X.500 Name Service via DIR.X, Relative Distinguished Names are returned in *service->h_host* and s*ervice->h_service*. Further details can be found in section “Requirements on communications applications” on page 42.

See also

uname(), t_getloc(), t_getaddr(); section “Requirements on communications applications” on page 42; chapter 6 “Configuration in Expert Mode” in manual “CMX, Operation and Administration” [1].

3.3.2 netdir_getbyaddr() - map a TRANSPORT ADDRESS to a GLOBAL NAME

This function is used to map a TRANSPORT ADDRESS to a GLOBAL NAME.

Synopsis

```
#include <netdir.h>
int netdir_getbyaddr(config, service, netaddr)
struct netconfig *config;
struct nd_hostservlist **service;
struct netbuf *netaddr;
```

Implementation-specific supplements

GLOBAL NAMES managed with the TNS have the following format:

Np5.Np4.Np3.Np2.Np1

Npi are name parts of the GLOBAL NAME in ascending hierarchical order, starting from left to right. If a name part is not used, but is followed by at least one name part that is used, the separator '.' must be inserted between the two name parts (a sequence of '.' separators at the end of the GLOBAL NAME can be omitted). If the '.' character is used within a name part, it must be specified as \. (backslash period).

After the *netdir_getbyaddr()* call, *netaddr* points to a *struct netbuf* containing a TRANSPORT ADDRESS. If the *netdir_getbyaddr()* function was successful, *(*service)->h_cnt* has the value 1.

*(*service)->h_hostservs->h_host* points to a character string that contains Np4.Np3.Np2.Np1, and *(*service)->h_hostservs->h_serv* points to a character string that contains Np5.

If the function is used to search in an X.500 Name Service via DIR.X, Relative Distinguished Names are returned in *(*service)->hostservs*. Further details can be found in section “Requirements on communications applications” on page 42.

See also

t_getname(); section “Requirements on communications applications” on page 42; chapter 6 “Configuration in Expert Mode” in manual “CMX, Operation and Administration” [1].

3.3.3 taddr2uaddr() - map a TRANSPORT ADDRESS to a universal address

This function is used to map a TRANSPORT ADDRESS to a universal address.

Synopsis

```
#include <netdir.h>
char *taddr2uaddr(config, addr)
struct netconfig *config;
struct netbuf *addr;
```

Implementation-specific supplements

taddr2uaddr() maps the TRANSPORT ADDRESS specified in *addr* to a universal address in accordance with the following rules:

– An octet whose value corresponds to the ASCII code of a printable character other than '\' is mapped to this character.

– An octet whose value corresponds to the ASCII code of the '\' character is mapped to the character string '\\'.

– An octet whose value corresponds to the ASCII code of the '\n' character is mapped to the character string '\n'.

– All other octets are mapped to the character string '\ddd', where ddd represents the octal value of the octet.

– The universal address is terminated with '\0'.

3.3.4 uaddr2taddr() - map a universal address to a TRANSPORT ADDRESS

This function is used to map a universal address to a TRANSPORT ADDRESS.

Synopsis

```
#include <netdir.h>
struct netbuf *uaddr2taddr(config, addr)
struct netconfig *config;
struct netbuf *addr;
```

Implementation-specific supplements

uaddr2taddr() maps the universal address specified in *addr* to a TRANSPORT ADDRESS in *struct netbuf* in accordance with the following rules:

– All characters apart from ’\’ are mapped to an octet whose value corresponds to the ASCII code of the character.

– The character string ’\\’ is mapped to an octet whose value corresponds to the ASCII code of the ’\’ character.

– The character string ’\n’ is mapped to the octet whose value corresponds to the ASCII code of the character ’\n’.

– The character string ’\ddd’ is mapped to the octet with the octal value ddd.

– The end character ’\0’ and all subsequent characters are ignored.

3.3.5 netdir_options() - interface to transport service options

Synopsis

```
#include <netdir.h>
int netdir_options(config, option, fd, pointer_to_args)
struct netconfig *config;
int option;
int fd;
char *pointer_to_args;
```

Implementation-specific supplements

Actions dependent on option:

ND_SET_BROADCAST
Not supported.

ND_SET_RESERVEDPORT
Not supported.

ND_CHECK_RESERVEDPORT
Not supported.

ND_MERGEADDR
If the function was successful,

((struct nd_mergearg)pointer_to_args)->m_uaddr* contains the same character string as that specified in

((struct nd_mergearg)pointer_to_args)->s_uaddr* before the call.

3.3.6 netconfig - network configuration file

Synopsis

```
#include <netconfig.h>
```

Implementation-specific supplements

As long as the product XTI V5.1 is installed, the network configuration file */etc/netconfig* contains three entries that describe the transport services provided by CMX≥V4.0 and the CCPs. The fields have the following values:

ISO transport services:

Network ID	**osicots**
Semantics	tpi_cots
Flag	-
Protocol family	osi
Protocol name	–
Network device file	t_osi_cots
Reference libraries	/usr/lib/tnsxaddr.so

Table 4: ISO transport services

NEA transport service:

Network ID	**nea**
Semantics	tpi_cots
Flag	-
Protocol family	nea
Protocol name	–
Network device file	t_neat
Reference libraries	/usr/lib/tnsxaddr.so

Table 5: NEA transport service

Message-oriented transport services:

Network ID	msg
Semantics	tpi_cots
Flag	-
Protocol family	msg
Protocol name	–
Network device file	t_msg
Reference libraries	/usr/lib/tnsxaddr.so

Table 6: Message-oriented transport services

See also

t_open()

3.4 Address management with DIR.X

3.4.1 Introduction

TNS address management enables communications applications to map symbolic names (GLOBAL NAMES) to protocol addresses (LOCAL NAMES and TRANSPORT ADDRESSES), and vice versa, by means of the function calls described in the sections “Access functions t_getaddr(), t_getloc(), and t_getname()” on page 19 and section “NETDIR access functions” on page 29. communications applications now have the additional option of obtaining this address information from DIR.X rather than TNS via the same program interface, provided a number of requirements are met.

DIR.X offers a distributed Directory Service via X.500. The term “distributed” means that the information sought is generally not located on the local system, but is stored on a remote system. However, this distribution of data is transparent to the communication application that requires the information. Detailed information on DIR.X can be found in the DIR.X description.

If you do **not** use DIR.X, you can skip this entire section. It does not contain any information on the installation, startup, and general administration of DIR.X, nor does it provide an introduction to X.500.

The section “The DIR.X Name Service” on page 39 contains a brief introduction to DIR.X, which should help you to understand the subsequent sections. The section “Requirements on communications applications” on page 42 describes the conditions under which a communications application can use DIR.X. Finally, the section “Parameterization of DIR.X” on page 44 describes how the system administrator should set the runtime environment of a communications application, so that the application can access DIR.X.

3.4.2 The DIR.X Name Service

Like TNS, DIR.X enables you to query addresses using symbolic names. Unlike TNS, however, this information is not stored locally, but on a particular server. The communications application sends a query to a Directory User Agent (DUA) via the one of the program interfaces described in the sections “Access functions t_getaddr(), t_getloc(), and t_getname()” on page 19 and “NETDIR access functions” on page 29. The DUA then sets up a Layer 7 (OSI) connection, known as an association, to a Directory Service Agent (DSA), and forwards the query to the DSA. The DSA searches its database and, if

successful, returns the desired information to the DUA, which passes it on to the communications application. If the DSA cannot provide the information itself, it reacts in one of three ways. If it has no information, it returns an error. If it knows of another DSA that may be able to provide the information, it either passes the name and address of the alternative DSA to the DUA, or requests the information from the alternative DSA itself. In first case, the DUA must send a new query to the second DSA; in the second case, the DUA need not take any further action.

These operations are transparent to the communications application. In certain circumstances, however, a DIR.X query may take longer than the corresponding TNS query. This is because, in extreme cases, several systems may have to be consulted. If queries are issued over public networks, this may incur a charge. Therefore, system and network administrators must be able to intervene, in order to optimize access to the DSA.

For instance, DIR.X provides a local cache in which the results of queries can be buffered and used again for a new query. The system administrator can define rules for using and updating the cache, and can thus directly influence costs. These and other options for controlling the behavior of DIR.X are described in more detail in section “Parameterization of DIR.X” on page 44.

The symbolic names used by TNS and DIR.X are basically similar, but differ in their syntax and semantics. In both cases, the complete name is made up of name parts, not unlike the structure of hierarchical name pools. In TNS, the complete name is known as a GLOBAL NAME, and consists of 5 name parts. Its syntax is NP5.NP4.NP3.NP2.NP1, where NPi (i = 1, 2, 3, 4, 5) represent the name parts. NP5 is the lowest level in the hierarchy, and NP1 is the highest level.

Example

```
Meier.Sales.Frankfurt.FSC.De
```

In DIR.X, the complete name is known as a Distinguished Name (DN). A DN also comprises a string of name parts, known as Relative Distinguished Names (RDN). Each DN can have a configurable number of name parts (max. 12 RDNs) with the following syntax:

```
RDN1[/RDN2/RDN3/.../RDNx]
```

The RDNs are separated by a slash (/); [ ]=optional.

An RDN consists of one or more attribute values, each of which comprises a type and a value. Common attribute types include the name of a country, an organization, or a person, etc.

DIR.X uses the following syntax for an RDN:

Syntax of an RDN:
attribute-value1[,attribute-value2,...,attribute-valuem]

Syntax of an attribute value:
type=value

[ ] optional

If an attribute value contains equals signs (=), commas (,), or slashes (/), these characters must be escaped by a preceding backslash (\).

An example of a DN in DIR.X notation is:

```
C=De/O=SNI/L=Frankfurt/OU=Sales/CN=Meier
```

The type identifiers are Common Name (CN), Organizational Unit (OU), Location (L), Organization (O), and Country (C). The DIR.X description indicates which type identifiers are predefined by DIR.X, and describes how to define your own type identifiers.

DNs are often quite long. DIR.X therefore offers the option of defining an alias name for a DN. The alias name is mapped to the correct DN in the local cache, which must be configured as appropriate by the DIR.X administrator. An alias name looks like a DN that contains a single RDN with the attribute ALI only.

Example

```
Alias name --> DN
ALI=Meier --> C=De/O=SNI/L=Frankfurt/OU=Sales/CN=Meier
```

If the DN of an alias name is not found when accessing the DSA, the alias entry is removed the next time the cache is updated.

3.4.3 Requirements on communications applications

Not all communications applications have been designed to work with DIR.X. The following rules must be observed when implementing the application:

- The functions t_getloc(), t_getaddr(), and t_getname() (see section "Access functions t_getaddr(), t_getloc(), and t_getname()" on page 19), and the functions netdir_getbyname(), and netdir_getbyaddr (see section "NETDIR access functions" on page 29) can be used.
- When using t_getloc() and t_getaddr(), symbolic names can be specified either as Distinguished Names or as GLOBAL NAMES. In the latter case, mapping rules for converting GLOBAL NAMES to Distinguished Names must be defined in the MAPRULES environment variable (see section "Facilities for the MAPRULES variable" on page 49). Please note, however, that these mapping rules apply to all applications started from a common UNIX shell. If different applications require different mapping rules, this leads to mapping conflicts.

  To ensure maximum flexibility, therefore, communications applications should handle symbolic names as simple character strings, and should not make any assumptions about their internal structure.
- When using netdir_getbyname(), mapping rules must always be defined in MAPRULES (see section "Facilities for the MAPRULES variable" on page 49), in order to map the character strings *service->h_host* and *service->h_serv* to a Distinguished Name.

  service->h_host is mapped using the NP4 facility, and *service->h_serv* using the NP5 facility. With netdir_getbyname(), therefore, only limited access to the X.500 name pool is possible.
- The t_getname() function can be used with the following restrictions and risks:
  - t_getname() always returns a complete Distinguished Name. The mapping rules between GLOBAL NAMES and Distinguished Names (see section "Facilities for the MAPRULES variable" on page 49) only apply to t_getloc() and t_getaddr(). Applications that require GLOBAL NAMES, therefore, cannot use t_getname().
  - t_getname() never returns alias names.
  - t_getname() initiates an X.500 search operation, which may be very time-consuming, and may result in high costs (line and usage fees).

- The netdir_getbyname() function can be used with the following restrictions and risks:
  - The character string *(*service)->h_hostservs->h_host* contains the RDN as defined by the NP4 facility mapping rules;

    *(*service)->h_hostservs->h_service* contains the RDN as defined by the NP5 facility (see section “Facilities for the MAPRULES variable” on page 49. All other RDNs of the DN are discarded.
  - netdir_getbyaddr() initiates an X.500 search operation, which may be very time-consuming, and may result in high costs (line and usage fees).
- The addresses returned by DIR.X may be longer than the addresses managed by TNS. Therefore, communications applications whose storage space is configured as the maximum length of a TNS address may not be able to use all addresses requested from DIR.X (see EA facility in section “Facilities for the NSCONTROL variable” on page 45).
- Queries sent to DIR.X using the functions described in sections “Access functions t_getaddr(), t_getloc(), and t_getname()” on page 19 and “NETDIR access functions” on page 29 are always processed as an anonymous user. This means that only PUBLIC information can be queried.
- Communications applications that use DIR.X must also observe the following convention:
  - all identifiers beginning with dx or dirx are reserved for DIR.X.
- If the BM facility (see section “Facilities for the NSCONTROL variable” on page 45) is used with the value TIMED:<mm>, the sigaction() function cannot be called while the specified timer is running. When the timer expires, any system calls of the communications application are interrupted by the SIGALRM signal.

3.4.4 Parameterization of DIR.X

To allow a communications application to access DIR.X, the system administrator must first set the correct runtime environment. This section assumes that DIR.X has already been installed and started; these steps are not described here.

Access to DIR.X is controlled by parameters; in the context of DIR.X, these are known as facilities. Each facility can be assigned one or more values with the following syntax:

<facility id>=<value1>[,<value2>,...,<valueN>]

[ ] = optional

<facility> represents the name of the facility, and <valuei> (i = 1, ..., N) represents a value. Each facility has a default value, which applies unless specified otherwise by the system administrator. If a number of facilities is set, the facility expressions are separated by semicolons (;):

<facility1>;<facility2>;...;<facilityN>

<facilityi> (i = 1, ..., N) represents an expression with the above syntax.

The facilities can be passed to the runtime environment in two ways:

- Definition of environment variables

  Facilities can be passed to the runtime environment using the NSCONTROL and MAPRULES environment variables.

  The general transfer syntax is:

  <environment variable>="<facility1>;...;<facilityN>";

  export_<environment variable>

  <environment variable> stands for NSCONTROL or MAPRULES. The exact assignment of environment variables to facilities is described in the following sections.

- Entry in /opt/lib/cmx/dirx.rc

  Facilities can be entered in the /opt/lib/cmx/dirx.rc file. This file must be created by the system administrator.

  The input syntax is:

  <configuration variable>=<facility1>;...;<facilityN>

  <configuration variable> stands for NSCONTROL or MAPRULES. Note that the quotes used in the environment variable syntax are omitted here.

  The assignment of configuration variables to facilities is described in the following sections.

  If only one facility is defined using the environment variable, the assignments in dirx.rc are ignored. It is not possible to mix the two transfer methods (environment variable or dirx.rc).

Example

Definition using an environment variable:

```
NSCONTROL="SI=DIR.X;RM=CACHE_ONLY;BM=TIMED:1"
MAPRULES="NP1=C;NP3=O;NP4=OU;NP5=CN"
export NSCONTROL,MAPRULES
```

Entry in /opt/lib/cmx/dirx.rc:

```
NSCONTROL=SI=DIR.X;RM=CACHE_ONLY;BM=TIMED:1
MAPRULES=NP1=C;NP3=O;NP4=OU;NP5=CN
```

The facilities are explained in the following sections.

3.4.4.1 Facilities for the NSCONTROL variable

SI Name Service selection facility

SI (Service Identifier) defines whether TNS or DIR.X is to be used as the Name Service. If SI is not set or SI=TNSX, TNS is accessed. In this case, all other facilities are ignored, because they control the behavior of DIR.X only.

The following values are possible:

TNSX
: TNS provides the Name Service

DIR.X
: DIR.X provides the Name Service

BM Bind mode facility

BM (Bind Mode) defines how often a Layer 7 connection (association) is set up to the DSA. To determine the optimum value, you must weigh the cost of using the public network against the time lost setting up and closing down connections plus the additional costs involved in connection setup. BM is significant only if SI=DIR.X.

The following values are possible:

<u>TMP</u>
: Temporary association. An association is set up each time the DSA is accessed, and closed down when access is terminated.

PERM
: Permanent association. An association is set up for an application process the first time the process accesses the DSA. When the process is terminated, the association is closed down.

TIMED:<mm>
: Timed association. An association is set up for accessing the DSA, provided the association does not already exist. Otherwise, the DSA is accessed via the existing association. If the DSA is not accessed within <mm> minutes, the association is closed down.

  For program restrictions with TIMED:<mm>, see section "Requirements on communications applications" on page 42.

RM Request mode facility

RM (Request Mode) enables you to select which databases are searched and the order in which they are searched following a service query. You can restrict the request to the local cache, to the adjacent DSA, or you can permit all X.500 databases that can be accessed by your DIR.X installation. The values are divided into three groups, which can be combined. If a value is not specified for one of the groups when defining RM, the default value for this group applies. If RM is not explicitly defined, the default values of all three groups apply.

RM is significant only if SI=DIR.X.

The following values are possible:

Group 1: Defining the query range

LOCAL_SCOPE
: The query refers only to the local cache and the DSA database to which the association has been set up (the adjacent DSA).

LOCAL_SCOPE,DSA_ONLY
: As for LOCAL_SCOPE, but excluding the local cache.

GLOBAL_SCOPE
: The query refers to the local cache and all X.500 databases that can be accessed by the DIR.X installation.

GLOBAL_SCOPE,DSA_ONLY
: As for GLOBAL_SCOPE, but excluding the local cache.

CACHE_ONLY
: The query refers to the local cache only.

Group 2: Defining the query order

CACHE_FIRST
: The local cache is searched first. This value is the group default, provided DSA_ONLY is set in group 1.

DSA_FIRST
: The DSA databases are searched first. This value is the group default, provided DSA_ONLY is set in group 1.

Group 3: Storage mode in the local cache

NORMAL_CLASS
: Results stored in the local cache can be overwritten by the next query issued to the DSA.

RESIDENT_CLASS
: Results are stored permanently in the local cache. They are not overwritten by the next DSA query.

DONT_STORE
: Results of the DSA query are not stored in the local cache.

Example

RM=LOCAL_SCOPE,CACHE_FIRST,RESIDENT_CLASS

The query refers to the local cache and the adjacent DSA. The local cache is searched first; if the desired information is not found here, a query is issued to the adjacent DSA. The DSA database is then searched, and the result stored permanently in the local cache.

The assignment contains two default values, and has the same meaning as RM=RESIDENT_CLASS. CACHE_ONLY,DSA_FIRST and DSA_ONLY,CACHE_FIRST cannot be combined.

EA Extended addressing

A communications application must provide storage space for the result of a Name Service query. The size of this storage space is generally the value

info->addr, which is supplied to the communications application as the result of a *t_open()* or *t_getinfo()* call (see chapter “Supplements to the function library” on page 51). This value is based on the maximum length of TNS addresses. A DIR.X query, however, may return a longer address. If a communications application is not prepared for this, reliable operation cannot be guaranteed. With EA (Extended Addressing), however, the system administrator can define whether addresses that exceed the maximum length are passed to the calling application, or suppressed. EA is significant only if SI=DIR.X.

The following values are possible:

<u>NO</u>

The communications application receives an error message if the result of the query is greater than “sizeof (union t_address)”.

YES

The result of the query is passed to the communications application without checking the length.

3.4.4.2 Facilities for the MAPRULES variable

These facilities enable TNS GLOBAL NAMES or host/service entries of the NETDIR interface to be mapped to DIR.X Distinguished Names. This mapping is necessary for migration reasons, if the communications application responds to Name Service queries by returning GLOBAL NAMES or host/service entries instead of Distinguished Names.

Mapping is carried our via the NP1, NP2, NP3, NP4, and NP5 facilities. By assigning a facility, the name part NPi (i = 1, ..., 5) of a GLOBAL NAME can be mapped to an RDN with a single attribute. The general syntax is NPi=<attribute type>, e.g. NP1=C (C represents the attribute type Country). <attribute type> can be empty, and “NPi=” is also permitted. This means that the name part NPi of a GLOBAL NAME is ignored during mapping.

If no facilities are set using MAPRULES, the following default value applies: MAPRULES=“NP1=C;NP2=;NP3=O;NP4=OU;NP5=CN”.

The attribute types are Country (C), Organization (O), Organization Unit (OU), and Common Name (CN). The attribute types recognized by DIR.X are given in the DIR.X description.

If MAPRULES is passed to the runtime environment with an empty value (MAPRULES=), no name mapping takes place.

When using the mapping mechanism, you must ensure that all GLOBAL NAMES used by the applications can be mapped to Distinguished Names.

If the application uses the NETDIR interface (see section “NETDIR access functions” on page 29), only the host and service entries are available. Host is mapped to an RDN via NP4, and service is mapped via NP5. Further information can be found in section “Requirements on communications applications” on page 42.

3.4.5 Mapping to local address formats

The addresses received by DIR.X are generally not in a format that can be understood by the local transport service providers, and must therefore be converted to local formats. The address information must also be supplemented by local information, e.g. the communication controller to be used to set up the connection. Both of these processes are carried out via a Network Directory Service (NDS) and are transparent to the communications application.

4 Supplements to the function library

This chapter supplements the description of the XTI function calls in the "X/Open Networking Services (XNS)" [5].

Functions implemented in XTI

All XTI functions listed in the "X/Open Networking Services (XNS)" [5] are implemented in XTI V5.1. Please note however that some of the functions are only supported by certain transport service providers. The functions used to support datagram communication, for example, require a connectionless transport service. If a function is not supported by a transport service provider, the function call fails with the variable *t_errno* set to [TNOTSUPPORT].

The table below contains all the functions implemented in XTI V5.1.

	ISO NEA	TCP	UDP
t_accept	X	X	
t_alloc	X	X	X
t_bind	X	X	X
t_close	X	X	X
t_connect	X	X	
t_error	X	X	X
t_free	X	X	X
t_getinfo	X	X	X
t_getprotaddr	X	X	X
t_getstate	X	X	X
t_listen	X	X	
t_look	X	X	X
t_open	X	X	X
t_optmgmt	X	X	X
t_rcv	X	X	
t_rcvconnect	X	X	

Table 7: Functions implemented in XTI V5.1

	ISO NEA	TCP	UDP
t_rcvdis	X	X	
t_rcvrel		X	
t_rcvreldata		X	
t_rcvudata			X
t_rcvv	X	X	
t_rcvuderr			X
t_rcvvudata			X
t_snd	X	X	
t_snddis	X	X	
t_sndrel		X	
t_sndreldata		X	
t_sndudata			X
t_sndv	X	X	
t_sndvudata			X
t_strerror	X	X	X
t_sync	X	X	X
t_sysconf	X	X	X
t_unbind	X	X	X

Table 7: Functions implemented in XTI V5.1

Supplements to the XTI function call descriptions

This section describes supplements to the following function calls:

t_accept()	t_rcvdis()
t_alloc()	t_rcvrel()
t_bind()	t_rcvudata()
t_connect()	t_rcvuderr()
t_error()	t_snd()
t_free()	t_snddis()
t_getinfo()	t_sndrel()
t_listen()	t_sndudata()

t_look()	t_sync()
t_open()	t_sysconf()
t_rcv()	t_unbind()
t_rcvconnect()	

As far as possible, the supplements follow the structure outlined below:

1. Each description starts with a repetition of the Synopsis of the XTI function call.
2. Implementation-specific supplements

   All peculiarities which should be observed when supplying the parameters in the application program are described under this heading. Subheadings indicate the parameter to which the supplement refers.
3. Notes

   Additional remarks concerning the X/Open definition of XTI are listed under this heading.
4. Supplements to error messages

   Under this heading those values of the *t_errno* variable are explained which may occur in the event of errors ignored by XTI or which are implementation-specific. The values of *errno* which may occur if *t_errno* = TSYSERR and their meaning are indicated.

4.1 t_accept() - accept a connect request

Synopsis

```
#include <xti.h>
int t_accept(fd, resfd, call)
int fd;
int resfd;
struct t_call *call;
```

Implementation-specific supplements

call->addr

The TRANSPORT ADDRESS which the communications application submits to XTI in *call->addr.buf* is ignored. In order to confirm connection establishment with *t_accept()* it is sufficient to specify the connect indication in *call->sequence*. *call->addr.len* and *call->addr.buf* can therefore be supplied with any values by the communications application.

Supplements to error messages

In addition to the meanings described in the “X/Open Networking Services (XNS)” [5], the errors indicated in *t_errno* may have the following meanings:

[TBADF]
: *resfd* and *fd* refer to two transport endpoints which are not bound to the same LOCAL NAME.

In the event of a system error, i.e. *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EINTR]
: The call was aborted by a signal.

[ENXIO] or [EIO]
: The transport service provider is no longer operable.

[EFAULT]
The area specified in *call->udata* or *call->opt* is not (or not completely) located in the user address space.

[EAGAIN] or [ENOSR]
XTI is temporarily unable to allocate any buffer area in main memory.

See also

t_bind()

4.2 t_alloc() - allocate a library structure

Synopsis

```
#include <xti.h>
char *t_alloc(fd, struct_type, fields)
int fd;
int struct_type;
int fields;
```

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have one of the following values:

[ENXIO] or [EIO]
: *t_alloc()* may have to procure the required information before it can allocate memory for a variable specified in *fields*. The [EIO] error message is returned if the transport provider is inoperable at this moment.

[ENOMEM]
: No memory area can be allocated since the system-specific limit value has already been exceeded.

4.3 t_bind() - bind an address to a transport endpoint

Synopsis

```
#include <xti.h>
int t_bind(fd, req, ret)
int fd;
struct t_bind *req;
struct t_bind *ret;
```

Implementation-specific supplements

req->addr

If automatic address generation is used, *req->addr* must be supplied by the communications application with a pointer to an area containing the LOCAL NAME of the communications application. The LOCAL NAME must agree with the transport service which the communications application specified in the *name* parameter of the *t_open()* call. Please also note the information contained in section “Transport services and LOCAL NAMES” on page 16.

The communications application can retrieve the LOCAL NAME from the TS directory with *t_getloc()* or *netdir_getbyname()* before *t_bind()* is called. The GLOBAL NAME of the communications application must be specified in the *t_getloc()* or *netdir_getbyname()* call. The communications application passes the LOCAL NAME returned by *t_getloc()* or *netdir_getbyname()* to *t_bind()* unchecked.

All transport service providers support automatic address generation.

Supplements to error messages

In addition to the meanings described in the “X/Open Networking Services (XNS)” [5], the errors indicated in *t_errno* may have the following meanings:

[TADDRBUSY]

The LOCAL NAME has already been assigned to a CMX application or an attempt was made with TCP/IP to bind two transport endpoints to the same local address.

[TBADADDR]
The LOCAL NAME does not agree with the transport service specified (see also supplement to *t_open()*) or a T-selector of the LOCAL NAME is also used in another LOCAL NAME bound to a different transport endpoint.

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[ENOMEM]
No further LOCAL NAME can be bound to a transport endpoint since the (system-dependent) limit value has been reached.

[EFAULT]
The area specified in *req->addr* or *ret->addr* is not (or not completely) located in the user address space.

[ENXIO] or [EIO]
The transport provider is no longer operable.

[ENOSR]
XTI is temporarily unable to allocate any buffer area in main memory.

See also

t_open(), t_accept(), t_getloc(), netdir_getbyname(), chapter “Address translation” on page 9.

4.4 t_connect() - establish a connection with another transport user

Synopsis

```
#include <xti.h>
int t_connect(fd, sndcall, rcvcall)
int fd;
struct t_call *sndcall;
struct t_call *rcvcall;
```

Implementation-specific supplements

sndcall->addr

sndcall->addr must be supplied by the communications application with a pointer to the TRANSPORT ADDRESS of the communication partner desired. The communications application can retrieve the TRANSPORT ADDRESS from the TS directory using *t_getaddr()* or *netdir_getbyname()* before *t_connect()* is called. The communications application must specify the GLOBAL NAME of the communication partner in the *t_getaddr()* or *netdir_getbyname()* call. The communications application passes the TRANSPORT ADDRESS returned by *t_getaddr()* or *netdir_getbyname()* to *t_connect()* unchecked.

sndcall->opt

The field *sndcall->opt.len* may be set to 0 if the standard options are to apply.

rcvcall

The TRANSPORT ADDRESS returned (in synchronous mode) upon successful completion of the function *t_connect()* in *rcvcall->addr* is always identical with the TRANSPORT ADDRESS which the communications application submitted to XTI in *sndcall->addr*.

If the maximum buffer length is set to 0 (*rcvcall->addr.maxlen, rcvcall->opt.maxlen, rcvcall->udata.maxlen*), the corresponding return information is discarded. Users therefore have the option of selecting specific return information.

If the function *t_connect()* runs successfully in synchronous mode when using TCP/IP, this means that the connection has been established, but the partner application has not yet necessarily accepted the connection with *t_listen()* / *t_accept()*.

Supplements to error messages

If the function *t_connect* aborts due to an error, the transport endpoint concerned retains the T_IDLE state. However, if a synchronous *t_connect* is interrupted by a signal after the connect request is sent and before the acknowledgment is received, the transport endpoint concerned switches to the T_OUTCON state. In addition to the meanings described in the “X/Open Networking Services (XNS)” [5], the errors indicated in *t_errno* may have the following meanings:

[TBADADDR]
: The TRANSPORT ADDRESS supplied does not agree with the transport service specified with *t_open()*.

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[ENOMEM]
: No further connection can be established since the (system-dependent) limit value has been reached or XTI was temporarily unable to allocate any buffer area in main memory.

[EFAULT]
: The area specified in *sndcall->addr, sndcall->udata*, *sndcall->opt, rcvcall->addr, rcvcall->udata* or *rcvcall->opt* is not (or not completely) located in the user address space.

[EINTR]
: The call has been interrupted by a signal.

[ENXIO] or [EIO]
: The transport provider is not (no longer) operable, or the CC list supplied with the TRANSPORT ADDRESS is incorrect.

[EAGAIN] or [ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

Other values that explain why the connection could not be established can occur with TCP/IP. These values are machine-specific and are defined and described in the file <sys/errno.h>.

t_connect() obtains information from the TNS as to which CC (Communication Controller) is used for communication. This may cause further systems errors which are not contained in this list.

See also

chapter “Address translation” on page 9, t_getaddr(), netdir_getbyname()

4.5 t_error() - produce error message

Synopsis

```
#include <xti.h>
int t_error(errmsg)
char *errmsg;
```

Implementation-specific supplements

In the following the format of the message texts output by *t_error()* is described. The output format described differs depending on whether *t_errno* has the value [TSYSERR] or any value other than [TSYSERR].

t_errno not equal to [TSYSERR]:
: First the message text of *errmsg* is output, provided that *errmsg* is not a null pointer and does not point to a null string. The message text is followed by a colon and a blank. Next a standard message text for the error indicated in *t_errno* is output, followed by a new line.

t_errno equal to [TSYSERR]:
: Again first the message text, followed by a colon and a blank, and then the standard message text referring to [TSYSERR] are output. This output is followed by a colon, a blank and the standard message for the error indicated in *errno*. This is followed by a new line.

t_error() makes use of the X/Open Native Language System (NLS) for the output of the standard messages. The language used for standard message output is defined by the value of the environment variable *LANG*.

The standard message texts are currently available in German and English. The standard English message texts correspond to the error comments given in *<xti.h>*. The message texts referring to system errors are identical to those of the function *perror()*.

See also

<xti.h>, perror(), “X/Open Networking Services (XNS)” [5]

4.6 t_free() - free a library structure

Synopsis

```
#include <xti.h>
int t_free(ptr, struct_type)
char *ptr;
int struct_type;
```

Caveat

The *t_free()* function should only be used to free memory previously allocated by *t_alloc()*. In particular, watch out for the following situation:

A structure has been allocated memory by means of *t_alloc()*. A member *buf* of this structure points to a memory area that was **not** allocated by *t_alloc()*. Freeing the memory using *t_free()* may then yield undefined results. This situation can be avoided by setting the *buf* member to NULL before freeing the memory with *t_free()*.

See also

t_alloc()

4.7 t_getinfo() - get protocol-specific service information

Synopsis

```
#include <xti.h>
int t_getinfo(fd, info)
int fd;
struct t_info *info;
```

Implementation-specific supplements

info

The function *t_getinfo()* uses the *info* pointer to supply the characteristics of the transport provider connected with the *fd* transport endpoint at the time the call is issued.

If one of the collective services *t_msg* or *t_osi_cots* was specified when opening the transport endpoint *fd* by *t_open()*, the *info* pointer of *t_getinfo()* returns the characteristics of the transport provider (or rather of the underlying transport protocol) which is used for the current transport connection. This means that the values returned for *t_osi_cots* can, for instance, refer to the characteristics of a transport protocol of class 0 or of a higher-level protocol class. An overview of the values that may be returned by *info* is given in the implementation-specific supplements to *t_open()*.

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]

The area specified in *info* is not (or not completely) located in the user address space.

[ENXIO] or [EIO]

The transport provider is no longer operable.

[ENOSR]

XTI is temporarily unable to allocate any buffer area in main memory.

See also

t_open()

4.8 t_listen() - listen for a connect request

Synopsis

```
#include <xti.h>
int t_listen(fd, call)
int fd;
struct t_call *call;
```

Implementation-specific supplements

call->addr

t_listen() uses the structure *call->addr* to return the TRANSPORT ADDRESS of the calling communications application. The TRANSPORT ADDRESS can be converted into the GLOBAL NAME of the calling communications application by means of the *t_getname()* or *netdir_getbyaddr()* call. If the maximum buffer length is set to 0

(*call->addr.maxlen, call->opt.maxlen, call->udata.maxlen*), the corresponding return information is discarded. Users therefore have the option of selecting specific return information.

If the function *t_listen* runs successfully when TCP/IP is used, this means that a complete connection, and not just a connection request, has been accepted. The connection must nonetheless be confirmed with *t_accept()* or rejected with *t_snddis()*.

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]

The area specified in *call->addr*, *call->udata* or *call->opt* is not (or not completely) located in the user address space.

[EINTR]

The call has been interrupted by a signal.

[EIO]
The transport provider is no longer operable.

[ENORS]
XTI is temporarily unable to allocate any buffer area in main memory.

See also

netdir_getbyaddr(), t_getname(), t_look, chapter “Address translation” on page 9.

4.9 t_look() - look at the current event on a transport endpoint

Synopsis

```
#include <xti.h>
int t_look(fd)
int fd;
```

Implementation-specific supplements

The return values have the following meanings:

T_LISTEN

(For TCP/IP). A connection to this transport endpoint has been **established**, not just **requested**. The connection must nonetheless be accepted with *t_listen()* / *t_accept()* or rejected with *t_listen()* / *t_snddis()*.

T_CONNECT

(For TCP/IP). The connection has been established. However, the partner application has not yet necessarily accepted the connection with *t_listen()* and *t_accept()*.

The data flow is restricted after a non-blocking t_snd() call, if the call is aborted with [TFLOW] or if the return value of t_snd() does not match the value passed to t_snd() in *nbytes*.

4.10 t_open() - establish a transport endpoint

Synopsis

```
#include <xti.h>
#include <fcntl.h>
int t_open(name, oflag, info)
char *name;
int oflag;
struct t_info *info;
```

Implementation-specific supplements

The *name* parameter can be used to specify one or more transport service providers.

t_open() checks whether at least one of the corresponding transport providers is available at the time the call is issued. The transport provider which matches the TRANSPORT ADDRESS of the communication partner is not selected from the CCPs corresponding to the transport service specified until the connection is established.

The permissible values of the *name* parameter and the characteristics of the transport services returned by *info* are listed below.

name

name points to a null-terminated character string identifying a transport service with specific characteristics. It can therefore be either a symbolic name of the form *t_xxx* or a device name. The names used are derived from the transport protocols providing the service.

XTI recognizes the following names:

t_neat

refers to the NEA transport service.

t_msg

Specifying this value for *name* enables the communications application to access any message-oriented transport service via the opened transport endpoint. The service *t_msg* is a collective service which includes the transport services *t_neat* and *t_osi_cots*. *t_open()* always establishes a transport endpoint when the service *t_msg* is specified, because at least local communication via the built-in loop-back provider is always possible.

Which of the transport services included under *t_msg* will actually handle the communication depends on the following:

- the CCP profiles activated at the time of connection establishment,
- the LOCAL NAME specified in the *t_bind()* call,
- the TRANSPORT ADDRESS of the remote communications application.

t_osi_cots
: Specifying this value for *name* enables the communications application to access any available ISO transport service via the opened transport endpoint. This includes the communication over RFC1006 (on top of TCP/IP). Which of the transport services will actually handle the communication depends on the same criteria as for *t_msg*.

t_rfc1006
: refers to a transport service on top of TCP/IP in accordance with ISO8072.

t_sinix
: refers to the transport service for local communication within the UNIX system. This service enables the TS application to establish a connection and exchange data with another TS application of the local system. (The TS applications need not be related!)

t_tp2
: refers to an ISO transport service based on a transport protocol of class 2. The transport protocol can be downgraded to class 0.

t_tp4
: refers to an ISO transport service based on a transport protocol of class 4.

/dev/*/tcp
: TCP/IP transport service. The full device name is specified in the Release Notice, since it is system-dependent.

/dev/*/udp
: UDP/IP transport service. The full device name is specified in the Release Notice, since it is system-dependent.

The association between these names and the CCPs providing the services is shown in the following table.

Transport service	**CCP profiles**
t_msg	WAN-NEA WAN-NX25 ISDN-CONS ISDN-NEA ISDN-NX25 WAN-CONS ISDN-CONS ETHN-CLNS Local communication RFC1006 on top of TCP/IP
t_neat	WAN-NEA WAN-NX25 ISDN-NEA ISDN-NX25
t_osi_cots	WAN-CONS ISDN-CONS ETHN-CLNS RFC1006 on top of TCP/IP
t_rfc1006	RFC1006 on top of TCP/IP
t_sinix	-
t_tp2	WAN-CONS ISDN-CONS
t_tp4	ETHN-CLNS
/dev/*/tcp	TCP/IP (part of the UNIX operating system)
/dev/*/udp	UDP/IP (part of the UNIX operating system)

Table 8: Association of transport services with CCP profiles

The transport service *t_sinix* does not require a CCP because it is a service for local communication. *Local communication* is the communication between two TS applications running on the same system.

The transport services *TCP/IP* and *UDP/IP* are provided by the UNIX operating system.

info

The values entered by XTI V5.1 in the *info* variable depend on the transport service specified in *name*. The table "Values of the members in info" on page 71 lists the possible values of the members in *info*. These values are of limited significance for the communications application. They merely indicate the maximum quality of service which the transport service has to offer. It is, however, possible that only a restricted quality of service is available when the transport connection is actually established. This can be illustrated by the following two examples:

1. The transport service *t_neat* is selected. The contents of the *tsdu* member in *info* indicate that a TSDU may have any length (*tsdu* =-1). However, this feature of the transport service cannot be used unless the transport connection on which it is provided links two UNIX systems. If, on the other hand, the transport connection exists with a communication computer under PDN or a host computer under BS2000/OSD, then the length of a TSDU is limited to 4096 bytes.
2. The collective service *t_msg* is selected. The highest-quality transport service available under *t_msg* is the one for local communication. The values for this transport service are returned in *info*. If the communications application subsequently establishes a transport connection using the NEA transport service, the values for *t_neat* apply to this transport connection. These values are given in the table below in the line for *t_neat*. As long as this transport connection exists, *t_getinfo()* will only return these values.

   These remarks apply analogously to the collective service *t_osi_cots*.

The following table contains the values valid for the individual transport services as returned by *t_open()* via *info*. The members in *info* which appear in the header line are explained in more detail in the section describing the *t_open()* function in the "X/Open Networking Services (XNS)" [5].

The table lists two alternative sets of characteristics for the *t_osi_cots* services. The lower-quality set is returned if the system configuration existing at the time the call is issued only permits communication via a class 0 transport protocol.

	info ->						
name value	addr	options	tsdu	etsdu	con-nect	dis-con	servtype
t_msg	136	1024	-1	16	32	64	T_COTS
t_neat	136	1024	-1	12	92	1	T_COTS
t_osi_cots	136 136	1024 1024	-1 -1	16 -2	32 32	129 64	T_COTS T_COTS
/dev/*/tcp	16	1024 (R.U.) 9264 (Sol.)	0	-1	-2	-2	T_COTS_ORD
/dev/*/udp	16	1024 (R.U.) 9408 (Sol.)	*)	-2	-2	-2	T_CLTS

Table 9: Values of the members in info

*) tsdu = see section “t_sndudata() - send a data unit” on page 87

/*/ system-specific part of the name, see Release Notice

-1 no restriction on the size of a TSDU

-2 the exchange of expedited data or sending of user data at connection establishment/release is not supported.

The values specified merely represent the maximum values.

Supplements to error messages

In addition to the meanings described in the “X/Open Networking Services (XNS)” [5], the errors indicated in *t_errno* may have the following meanings:

[TBADNAME]
: Symbolic name of the form t_xxx: No transport provider can be found which bears the (syntactically correct) name (*name*): either the name does not exist in the conversion table used by the access system to determine the transport provider, or the conversion table was not loaded at system startup.

  Device name: No special file with the specified name exists.

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The memory reserved with *name* is not (or not completely) located in the user address space.

[ENXIO] or [EIO]
: There is no matching transport provider or the transport provider is not operable.

[ENOMEM]
: The maximum number of transport endpoints permitted by the system has already been reached. No more transport endpoints can be established at this moment.

[ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

See also

t_bind()

4.11 t_rcv() - receive data or expedited data sent over a connection

Synopsis

```
#include <xti.h>
int t_rcv(fd, buf, nbytes, flags)
int fd;
char *buf;
unsigned nbytes;
int *flags;
```

Implementation-specific supplements

Maximum performance of the access system is achieved if the buffer pointed to by *buf* is sufficiently large to hold an entire TSDU or ETSDU.

An error causing the *t_rcv()* function to abort may result in data being lost.

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The memory specified with *buf* and *nbytes* is not (or not completely) located in the user address space. Receive data are lost.

[EINTR]
: The call has been interrupted by a signal.

[EIO]
: The transport provider is no longer operable (at least for this transport endpoint).

[ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

4.12 t_rcvconnect() - receive the confirmation from a connect request

Synopsis

```
#include <xti.h>
int t_rcvconnect(fd, call)
int fd;
struct t_call *call;
```

Implementation-specific supplements

call->addr

Upon successful completion, the address returned in *call->addr* is identical with the TRANSPORT ADDRESS of the communication partner previously submitted by the communications application in *sndcall->addr* of *t_connect()*. When required, the communications application can obtain the GLOBAL NAME of the communication partner with this TRANSPORT ADDRESS from the TS directory by calling *t_getname()* or *netdir_getbyaddr().* If the maximum buffer length is set to 0

(*call->addr.maxlen, call->opt.maxlen, call->udata.maxlen*), the corresponding return information is discarded. Users therefore have the option of selecting specific return information.

If the function *t_rcvconnect()* runs successfully when TCP/IP is used, this means that the connection has been established; however, the partner application has not yet necessarily accepted the connection with *t_listen()* / *t_accept()*.

Supplements to error messages

If an error is indicated which sets *t_errno* = [TSYSERR] and *errno* = [EFAULT] or [EIO], this means that the connect confirmation has not become effective i.e. that connection establishment could not be completed. The *t_rcvconnect()* call cannot be repeated in this case, because it would again be aborted with *t_errno* = [TSYSERR] and reason [EIO]. Instead *t_close()* should be called to recover the error.

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The area specified in *call->addr* or *call->udata* is not (or not completely) located in the user address space.

[EINTR]
: The call has been interrupted by a signal.

[EIO]
: The transport provider is no longer operable (at least for this transport endpoint).

[ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

See also

t_connect(), t_getname(), netdir_getbyaddr(), chapter “Address translation” on page 9.

4.13 t_rcvdis() - retrieve information from disconnect

Synopsis

```
#include <xti.h>
int t_rcvdis(fd, discon)
int fd;
struct t_discon *discon;
```

Implementation-specific supplements

discon

If the *udata.maxlen* field is set to 0, any incoming user data will not be passed to the communications application. Upon successful completion, the function terminates with a return value of 0, and *discon->reason* contains the reason for connection release in decimal representation.

If the communications application is to be portable and protocol-independent, it should not interpret the values supplied in *discon->reason*.

When TCP/IP is used, the reason for connection release corresponds to one of the error numbers defined and described in the machine-specific file *<sys/errno.h>*.

The list below gives the possible reasons for connection release with ISO and NEA transport services as well as the meaning of these reasons. The reasons marked with an asterisk (*) are defined in ISO 8073. All other reasons are implementation-specific.

Reasons for connection release

Reasons for connection release originating from the transport providers:

Code	Reason
0*	Remote disconnect, no reason specified possibly due to user error of partner
1*	Remote disconnect due to TSAP congestion
2*	Remote disconnect due to not established TSAP
3*	Remote disconnect due to unknown TSAP

Table 10: Reasons for connection release

Code	Reason
5	Remote disconnect by (network) administration
6	Error in network
128*	Normal disconnect by the partner communications application
129*	Remote disconnect due to congestion during connection establishment
130*	Remote disconnect due to failure in connection negotiation
131*	Remote disconnect due to detection of duplicate source reference for the same pair of NSAPs
132*	Remote disconnect due to mismatched references
133*	Remote disconnect due to protocol error
135*	Remote disconnect due to reference overflow
136*	Remote disconnect due to connection refusal on this network connection
138*	Remote disconnect due to invalid header or parameter length
192	Local disconnect due to congestion
193	Local disconnect, QoS can no longer be provided
195	Local disconnect due to invalid (connection) password
196	Local disconnect due to denial of network access
208	Local disconnect due to protocol error
209	Local disconnect, received too long TIDU
210	Local disconnect due to violation of flow control for normal data
211	Local disconnect due to violation of flow control for expedited data
212	Local disconnect due to invalid TSAP id
213	Local disconnect due to invalid TCEP id
214	Local disconnect due to invalid parameter
224	Connection inhibited by local administration
225	Disconnect by local administration
226	Connection can not be set up locally because no network connection can be established
227	Local disconnect due to loss of network connection

Table 10: Reasons for connection release

Code	Reason
228	Connection can not be set up because partner does not respond to CONRQ
229	Local disconnect due to loss of connection (Idle Traffic Inactivity Timeout)
230	Local disconnect because resynchronization failed (more than 10 retries)
231	Local disconnect because expedited channel is inoperable

Table 10: Reasons for connection release

Reasons for connection release originating from XTI

Code	Reason
258	Local release by XTI due to deactivation of the CCP by administration
259	Local release by XTI due to failure of the CCP

Table 11: Reasons for connection release (XTI)

Supplements to error messages

If an error is indicated which sets *t_errno* = [TSYSERR] and *errno* = [EFAULT] or [EIO], user data, if any, and the reason for connection release

(*discon->reason*) are lost. The *t_rcvdis()* call cannot be repeated in this case, because it would again be aborted with *t_errno* = [TSYSERR] and reason [EIO]. Instead *t_close()* should be called to recover the error.

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The area specified in *discon->udata* is not (or not completely) located in the user address space.

[EINTR]
: The call was aborted by a signal.

[EIO]
: The transport provider is no longer operable (at least for this transport endpoint).

[ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

4.14 t_rcvrel() - acknowledge receipt of an orderly release indication

Synopsis

```
#include <xti.h>
int t_rcvrel(fd)
int fd;
```

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EINTR]
The call was aborted by a signal.

[ENOSR]
XTI is temporarily unable to allocate any buffer area in main memory.

4.15 t_rcvudata() - receive a data unit

Synopsis

```
#include <xti.h>
int t_rcvudata(fd, unitdata, flags)
int fd;
struct t_unitdata *unitdata;
int *flags;
```

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The area specified in *unidata->addr, unitdata->udata* or *unidata->opt* is not (or not completely) located in the user address space.

[EINTR]
: The call was aborted by a signal.

[ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

4.16 t_rcvuderr() - receive a unit data error indication

Synopsis

```
#include <xti.h>
int t_rcvuderr(fd, uderr)
int fd;
struct t_uderr *uderr;
```

Implementation-specific supplements

uderr->error corresponds to one of the error numbers defined and described in the machine-specific file *<sys/errno.h>*.

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The area specified in *uderr->addr or uderr->opt* is not (or not completely) located in the user address space.

[EINTR]
: The call was aborted by a signal.

[ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

4.17 t_snd() - send data or expedited data over a connection

Synopsis

```
#include <xti.h>
int t_snd(fd, buf, nbytes, flags)
int fd;
char *buf;
unsigned nbytes;
int flags;
```

Implementation-specific supplements

flags

XTI allows the communications application to concatenate expedited data by means of T_MORE when passing the data to XTI. It is advisable not to use this feature, since transport providers based on the ISO or NEA protocol cannot accept and transfer ETSDUs except as single units. Therefore, if a communications application submits expedited data to *t_snd()* with the T_MORE flag set, the expedited data will be locally buffered by XTI until the last expedited data (with T_MORE no longer set) has been submitted. It is not until then that XTI passes the expedited data to the transport system. Normal data flow is not affected by this. If the T_MORE flag is set when sending expedited data, it may occur that XTI does not notify the user until the last *t_snd()* call is issued whether the amount of data submitted exceeds the maximum permissible or not. If the limit value is exceeded *t_errno* is set to [TBADDATA]. The message is not sent in this case, and **all** the data comprising the message must be submitted again.

In non-blocking mode (O_NONBLOCK set) *t_snd()* can terminate successfully without having sent off all *nbytes* of data. The return value in this case indicates the number of data items already transferred. If the return value is smaller than *nbytes*, the remaining data items can be transferred by the communications application issuing another *t_snd()* call.

In the event of an error, *t_snd()* returns a value of -1. This means that there is no indication of the number of data items already sent up to this point!

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The area specified in *buf* and *nbytes* is not (or not completely) located in the user address space. Part of the data may have been sent. If the T_MORE flag is set this indicates to the receiving partner that the TSDU is incomplete.

[EINTR]
: The call has been interrupted by a signal. Part of the data may have been sent. If the T_MORE flag is set this indicates to the receiving partner that the TSDU is incomplete.

[ENXIO] or [EIO]
: The transport provider is no longer operable (at least for this transport endpoint). Part of the data may have been sent. If the T_MORE flag is set this indicates to the receiving partner that the TSDU is incomplete.

[ENOSR] or [ENOMEM]
: XTI was temporarily unable to allocate any buffer area in main memory to hold the data. No data has been sent.

4.18 t_snddis() - send user-initiated disconnect request

Synopsis

```
#include <xti.h>
int t_snddis(fd, call)
int fd;
struct t_call *call;
```

Implementation-specific supplements

If the function is aborted with [TSYSERR] and the reason [EFAULT], [ENXIO] or [EIO] specified in *errno*, the connection is still not considered to be released by XTI. The disconnect request may nevertheless have been sent.

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The area specified in *call->udata* is not (or not completely) located in the user address space.

[EINTR]
: The call is aborted by a signal.

[ENXIO] or [EIO]
: The transport provider is no longer operable (at least for this transport endpoint).

[EAGAIN] or [ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

4.19 t_sndrel() - initiate an orderly release

Synopsis

```
#include <xti.h>
int t_sndrel(fd)
int fd;Supplements to error messages
```

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EINTR]
The call is aborted by a signal.

[ENXIO] or [EIO]
The transport provider is no longer operable (at least for this transport endpoint).

[EAGAIN] or [ENOSR]
XTI is temporarily unable to allocate any buffer area in main memory.

4.20 t_sndudata() - send a data unit

Synopsis

```
#include <xti.h>
int t_sndudata(fd, unitdata)
int fd;
struct t_unitdata *unitdata;
```

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[EFAULT]
: The area specified in *unidata->addr, unitdata->udata* or *unidata->opt* is not (or not completely) located in the user address space.

[EINTR]
: The call is aborted by a signal.

[ENXIO]
: The transport provider is no longer operable (at least for this transport endpoint).

[ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

As a result of implementation-specific supplements to the TCP/IP transport service, the maximum size of the TSDU may be less than the value specified in *info->tsdu* in certain UNIX versions.

During operation via UDP/IP: *t_sndudata()* can only transfer a maximum of one TSDU as large as a STREAMS message. The maximum size of such a message is a configurable system parameter:
Reliant UNIX: *STRMSGSZ* in */etc/conf/cf.d/mtune* or */etc/conf/cf.d/stune*
Solaris: */etc/system*
command *sysdef* shows the actual value

4.21 t_sync() - synchronize transport library

Synopsis

```
#include <xti.h>
int t_sync(fd)
int fd;
```

When using the TCP/IP protocol, please note that the current values for *qlen* and *ocnt* are not inherited by the child process when the *fork()* system call is executed, as a result of the implementation. If one of the two processes is to continue to wait for incoming connection requests after the *fork()* call, then this must always be the parent process.

4.22 t_sysconf() - get configurable XTI variables

Synopsis

```
#include <xti.h>
int t_sysconf(int name);
```

Implementation-specific supplements

The only variable currently supported is variable -SC_T_IOV-MAX. The variable specifies the maximum number of non-contiguous buffers used for scatter/gather transfer (see functions t_sndv(), t_rcvv(), t_sndvudata(), t_rcvvudata(), t_rcvreldata(), t_sndreldata() and t_sysconf() in the “X/OPEN Networking Services (XNS)” [5]. The current value of the variable is 16.

4.23 t_unbind() - disable a transport endpoint

Synopsis

```
#include <xti.h>
int t_unbind(fd)
int fd;
```

Supplements to error messages

In the event of a system error, i.e. if *t_errno* has the value [TSYSERR], the system variable *errno* may have any of the following values:

[ENXIO]
: The transport provider is no longer operable (at least for this transport endpoint).

[ENOSR]
: XTI is temporarily unable to allocate any buffer area in main memory.

5 Supplements to the options

This section describes the supplements to the *t_optmgmt* call, which is used to manage the options of a transport endpoint.

5.1 t_optmgmt() - manage options for a transport endpoint

Synopsis

```
#include <xti.h>
int t_optmgmt(fd,req,ret)
int fd;
struct t_optmgmt *req;
struct t_optmgmt *ret;
```

Implementation-specific supplements

Currently supported options are:

For *t_opthdr.level* == XTI_GENERIC:

– XTI_DEBUG

For *t_opthdr.level* == ISO_TP:

– TCO_EXPD

For *t_opthdr.level* == INET_TCP:

– TCP_KEEPALIVE, TCP_MAXSEG, TCP_NODELAY

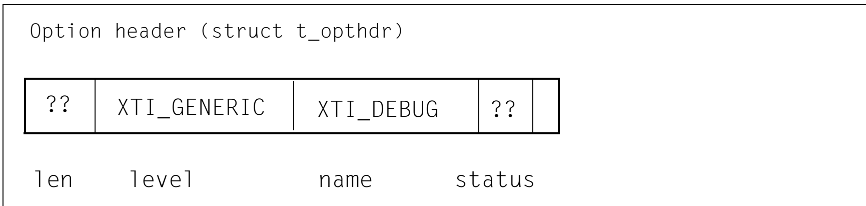
For *t_opthdr.level* == INET_IP:

– IP_BROADCAST, IP_DONTROUTE, IP_OPTIONS, IP_REUSEADDR

All options are available to all users, i.e. there are no privileged users.

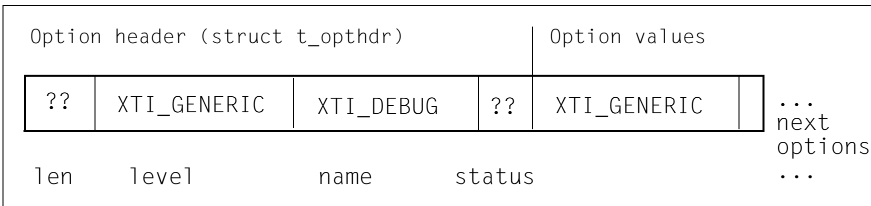
The option XTI_DEBUG of the XTI_GENERIC level has a different meaning depending on the option values attached to it. Option values can be specified as an array of type *long*. Basically, there are 3 valid formats of this option:

Format 1:



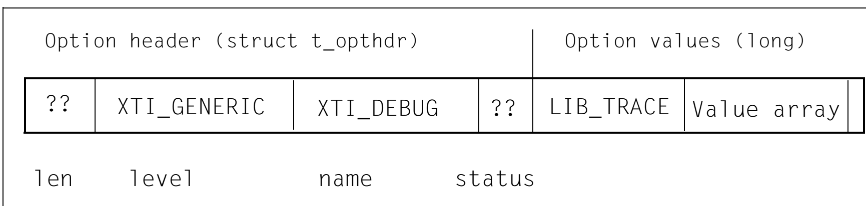
If there are no option values specified at all, the XTI library trace is disabled.

Format 2:



If XTI_GENERIC is the first member in the option value array, the XTI library trace is enabled with a default set of trace options. Subsequent option values are ignored.

Format 3:



If LIB_TRACE is the first member in the option value array, the XTI library trace is enabled, depending on the subsequent option values in the array. All values that can be specified for the XTITRACE environment variable are permitted, apart from the *-f* option (see chapter “XTI library trace” on page 101 for more detailed information).

If the trace parameters are to be changed while the trace is active, you must first explicitly disable the trace, and then restart it with the new parameters.

Use the LIBTR_ASSIGN macro to assign the desired values to the option value array:

```
LIBTR_ASSIGN (position, parameter, value)
                             |         |          |
                             |         |        Parameter value if set, and
                             |         |       'r'; otherwise 0
                             |         |       (Type: short)
                             |
                             |        Trace parameter: 's', 'S', 'r', or 'p'
                             |                          (Type: char)
                             |
                            Address of the member within the
                                                   option value array
                                                 (Type: pointer to long)
```

e.g.

LIBTR_ASSIGN(position,'S',0) corresponds to XTITRACE=-S,
LIBTR_ASSIGN(position,'s',0) corresponds to XTITRACE=-s, and
LIBTR_ASSIGN(position,'r',1000) corresponds to XTITRACE="-r 1000".

When the options are returned (e.g. T_optmgmt() with req->flags = T_CURRENT) with the option name = XTI_DEBUG and LIB_TRACE as the first member in the subsequent option value array, the user can load the option values set using the LIBTR_READ macro.

```
LIBTR_READ (position, parameter, value)
                           |          |           |
                           |          |    Field for returning the param. value
                           |          |              (Type: pointer to short)
                           |          |
                           |
                           |        Field for returning the trace parameter
                           |                           (Type: char)
                           |
                            Address of the member within the
                                                   option value array
                                                 (Type: pointer to long)
```

i Although the *t_optmgmt()* call is TEP-specific, the library trace enabled by the call is specific to a particular process (as when activated by the XTITRACE variable).

Supplements to error messages

[TBADOPT]
reg->flags == T_CHECK must not be combined with the T_ALLOPT option name.

[TOUTSTATE]
Cannot occur, because *t_optmgmt()* is permitted in all states except for T_UNINIT. In this state, however, there are no valid file descriptors, which means that [TBADF] would be returned.

6 Supplements to event management

Event management as defined by X/Open and implemented in XTI V5.1 always refers to a particular transport endpoint (fd in *t_look()*). A communications application cannot therefore simultaneously wait for different, asynchronous events at various transport endpoints. The definition of a general interface for event management has not yet been finalized.

You can however use system-specific interfaces, which allow communications application processes to expect data on various transport connections, for example, or to simultaneously monitor transport connections and standard input. Different system interfaces are available for this purpose in the different operating systems: *poll()* and *select()*.

A **single** *poll()* call can be used not only to control various transport endpoints, but also to simultaneously control entities that can be accessed via the poll mechanism, e.g. keyboard drivers.

A description of the function calls is given in the manual “CMX, Programming Applications” [2] that describes the C development system for your system. A list of the event management system calls that can be used on your end system is given in the Release Notice.

6.1 poll() - multiplex input and output entities

Different event bits must be set in *events* depending on the XTI event expected.

The following table shows the relationships between XTI events and *poll()* bits:

Event class and XTI event	**Event bits in „events“**
1 T_CONNECT	POLLIN or POLLRDNORM
1 T_DISCONNECT	POLLIN or POLLRDNORM
1 T_LISTEN	POLLIN or POLLRDNORM
1 T_ORDREL	POLLIN or POLLRDNORM
1 T_UDERR	POLLIN or POLLRDNORM
1 T_DATA	POLLIN or POLLRDNORM
1 T_EXDATA	POLLIN or POLLRDBAND
2 T_GODATA	POLLOUT or POLLWRNORM
2 T_GOEXDATA	POLLWRBAND

Table 12: Relationships between XTI events and poll() bits

For portability reasons, the following event bits must be set for applications that are designed to run on non-UNIX systems also:

Event class and XTI event	**Event bits in „events“**
1	POLLIN | POLLRDNORM | POLLRDBAND | POLLPRI
2	POLLOUT | POLLWRBAND

Table 13: Event bits to be set for applications running also on non-UNIX systems

If *t_look()* is then called for this transport endpoint, *fd* indicates which XTI event has occurred (e.g. T_DATA or T_EXDATA).

If a bit is set in *events* for a class 2 event, and if there is no send congestion for the relevant flow (normal data or expedited data), then *poll()* immediately returns successfully.

6.2 select() - multiplex file descriptors

The table below indicates the areas where a bit must be set in order that the respective event can be checked for the corresponding transport endpoint.

Event class and XTI event	Bit to be set in		
	readfds	**writefds**	**exceptfds**
1 T_CONNECT 1 T_DISCONNECT 1 T_LISTEN 1 T_ORDREL 1 T_UDERR 1 T_DATA 1 T_EXDATA	x x x x x x		x
2 T_GODATA 2 T_GOEXDATA		x	

Table 14: Event class and event bits at select()

For portability reasons, bits must be set in the following areas for applications that are designed to run on non-UNIX systems also:

Event class andXTI event	Bit to be set in		
	readfds	**writefds**	**exceptfds**
1	x		x
2		x	

Table 15: Event class and event bits at select() on non-UNIX systems

The flow of expedited data in the send direction cannot be monitored with *select()*.

7 Read/write interface

An interface is provided for communications applications that access transport endpoints using the system calls *read()* and *write()* as described in chapter 2 “Sockets Interfaces” of the “X/Open Networking Services (XNS)” [5]. This section provides information on how to use *read()* and *write()* when the specified file descriptor is a transport endpoint. You must be familiar with these functions.

7.1 read() - read from a file

Synopsis

```
#include <sys/types.h>
#include <unistd.h>
int read(fd, buf, nbyte)
int fd;
void *buf;
unsigned nbyte;
```

Supplements to error messages

In the event of error, *errno* is set to one of the following:

[EIO]
: The TSP is no longer operable.

[EFAULT]
: The area specified in *buf* and *nbyte* is not (or not completely) located in the user address space.

[EAGAIN]
: *fd* is accessed in non-blocking mode, and there is currently no receive data.

7.2 write() - write to file

Synopsis

```
#include <sys/types.h>
#include <unistd.h>
int write(fd, buf, nbyte)
int fd;
void *buf;
unsigned nbyte;
```

Supplements to error messages

In the event of error, *errno* is set to one of the following:

[EIO]
The TSP is no longer operable.

[EFAULT]
The area specified in *buf* and *nbyte* is not (or not completely) located in the user address space.

[EAGAIN]
fd is accessed in non-blocking mode and no send data can be accepted at present.

[ENXIO]
The transport connection was aborted (under normal circumstances by the partner application).

8 XTI library trace

The XTI library trace is controlled by means of the environment variable XTITRACE. This environment variable is used to switch on the trace and determine the amount of information to be collected. Alternatively, the trace can be activated at runtime using the *t_optmgmt()* function (see chapter “Supplements to the options” on page 91).

The trace entries are then collected in a ring buffer and saved in temporary files. These files are processed separately by the program *xtil*. The extent of the evaluation is determined by specifying particular options when calling the program *xtil*.

The options available when defining the environment variable XTITRACE, as well as the options that can be selected when calling *xtil*, are described in the following sections.

Notational conventions (XTI library trace)

The following notational conventions are used:

bold
    Constants; enter what appears in this manual

normal print
    Positional operands for which you can enter selected options or values

[ ]   May be omitted. The brackets must not be entered

␣   Mandatory blank

...   Previous expression may be repeated any number of times

8.1 XTITRACE - control the trace

The first XTI call issued by a process evaluates the environment variable XTITRACE and switches on the trace if necessary. After the trace is switched on, the temporary file *XTIF<pid>* with the process ID *<pid>* is opened, if it is not already open.

When the file *XTIF<pid>* has been written up to the maximum length, the entries are written to the file *XTIS<pid>*. When this second file is full, the first file *XTIF<pid>* is reverted to. This file is first cleared and then written. The directory is specified in the Release Notice.

The access rights `rw-------` (0600) are assigned to the files and can be found under the user ID of the process. Memory is then dynamically allocated for buffering the trace entries. The memory and files remain allocated for the duration of the process.

The options specified in XTITRACE control the trace mechanism. The options *s* and *S* determine the range of information recorded, while the options *p*, *r*, and *f* control the buffering, cyclical overwriting, and storage of the file respectively.

XTITRACE="-option1[␣**-p**␣fac][␣**-r**␣wrap][␣**-f**␣dir]";

export␣XTITRACE

Meaning of the options and parameters:

-option1

option1 determines the type of trace. Only one of the two possible values for *option* may be specified. In order to activate the trace, a value **must** be specified here.

The following alternatives can be specified for *option1*:

s The function names, the values of the arguments and the return values of the functions are logged. If an error occurs, *t_errno*, *errno* and the error position *errpos* are output in the library.

S The same information is logged as for option *s*. In the case of arguments that are pointers, the data structures addressed by the pointers are also logged. In practice, the *S* option should be used in preference to the *s* option.

-p␣fac

The decimal number *fac* determines the buffering factor. The size of the buffer is *fac* * BUFSIZ, where BUFSIZ is specified in *<stdio.h>*.

If *fac* > 8, the value for *fac* is automatically reduced to 8.
If *fac* = 0, each trace entry is written directly to the file (unbuffered).

-p␣fac not specified: *fac* = 1 assumed.

-r␣wrap

The decimal number *wrap* specifies that after *wrap* * BUFSIZ bytes (BUFSIZ specified in <*stdio.h*>), the trace entries are logged in the second temporary file *XTIS<pid>*.

This file handles the trace information in exactly the same way as *XTIF< pid>*. After every *wrap* * BUFSIZ bytes, the trace mechanism switches between *XTIF<pid>* and *XTIS<pid>*, whereby the old contents of the respective file are overwritten.

-r␣wrap not specified: *wrap* = 512 assumed.

-f␣dir

The *dir* entry specifies the directory in which the trace files *XTIF<pid>* and *XTIS<pid>* are to be stored.

-f␣dir not specified: The default directory for trace files is used; the name of this directory is specified in the Release Notice.

8.2 xtil - edit the trace information

xtil reads the entries generated by the trace from the temporary file *file*, processes them in accordance with the options specified, and outputs the result to *stdout*.

If the program runs successfully, the end status is 0, otherwise it is a value not equal to 0.

The format of *xtil* output is described in the next section.

xtil[␣-option2]␣file ...

Meaning of the options and parameters:

-option2

The options specified for *option2* determine which trace entries from *file* are to be edited. More than one of the values described below for *option2* can be specified in each *xtil* call.

-option2 not specified: *option2 = cdm* assumed.

The following values can be specified for *option2*:

c The trace entries are edited for XTI calls:

- for attaching/detaching the communications application
- for connection establishment/release

These calls include *t_accept(), t_bind(), t_close(), t_connect(), t_listen(), t_open(), t_rcvconnect(), t_rcvdis(), t_rcvrel(), t_snddis(), t_sndrel()* and *t_unbind()*.

d The trace entries are edited for XTI calls for data exchange. These calls include *t_rcv(), t_rcvudata(), t_rcvuderr(), t_snd()* and *t_sndudata()*.

m The trace entries are edited for the remaining XTI calls not edited with the options *c* and *d*. These calls include *t_alloc(), t_error(), t_free(), t_getaddr(), t_getinfo(), t_getloc(), t_getname(), t_getstate(), t_look(), t_optmgmt()* and *t_sync()*.

v The XTI calls, their arguments and options are edited in full. In the case of arguments passed as pointers, the data structures addressed are also output. The extent to which the data is edited depends on the options specified in XTITRACE. If the trace was enabled with the *S* option, the *v* option is recommended for editing. If only *v* is specified for *option2*, this has the same meaning as *option2 = cdmv*.

file ...
Name of one or more files with binary trace entries that are to be edited.

Output format of the XTI library trace

The trace information edited by *xtil* always begins with the following header:

```
XTI    TRACE (Vx.x)              Fri Aug 12 15:13:34 1990
OPTIONS 'cdmv' , TRACE FILE 'XTIF00963'
```

These two lines are output once before the trace data is output and contain the following information:

- Version of the XTI function library,
- Start date and start time of the trace,
- The selected editing options,
- Name of the edited trace file.

The trace information for the individual XTI calls is output in several lines with different formats. The extent of the output depends on the options specified for XTITRACE and *xtil*.

The first line of output always appears. It contains the following information:

A time stamp appears at the start of the first line in the following form:

```
<minutes>:<seconds>.<milliseconds> (e.g. 24:16.320).
```

The accuracy of the milliseconds specification depends on the type of machine.

This is followed by the XTI call logged *(t_xxxxx)* and, in parentheses, the arguments and their values in the order required by XTI. The arguments appear in decimal (%d), hexadecimal (0x%x), or symbolic (%s) form. In hexadecimal representation, 0x precedes the argument.

The following must be noted when interpreting the logged values:

– In the case of arguments that are addresses, the argument is represented in the form (0x%x).

– In the case of arguments of type integer (*.len, *.maxlen), the corresponding value is represented in the form 0x%x, %d, or %s. It is separated from the argument name by a blank.

For functions whose processing depends on the file mode, it is also specified whether the access is blocking (specification: BLOCK) or non-blocking (specification: NBLOCK).

The following lines are only output if the option *v* was specified for *xtil* and the trace has collected appropriate information (option *S* with XTITRACE).

In the case of arguments that are pointers, these lines also contain the data structures addressed by these pointers. The values of the structure components are converted into hexadecimal format and into plain text. The naming conventions used for arguments and structure components correspond to the specifications in the “X/Open Networking Services (XNS)” [5].

The following applies for identifying structure components:

>	The component must be assigned a valid value by the communications application.
<	The component is assigned a valid value by the XTI function if the function runs without error.
-	The value of the components is of no significance for the logged XTI call.
---	If --- is specified instead of the component value, the component is not assigned a valid value.

The last line of the output for an XTI call contains the return value. If an error occurs, *t_errno*, possibly *errno*, and information on the error position (*errpos*) are output in the library.

Example

Below is an example of detailed logging for an XTI call.

```
24:16.320 t_bind (fd  5, req 0x8054ac8, ret  0x0)
       req:        addr.maxlen(-)  addr.len(>)     addr.buf(>)
                   ---             24              0x8054d48
          0  01001800 0e000000  00000004 04003234|      24|
         10  39370000 00000000                   |97      |
                     qlen (>) 10
        return: 0
```

Glossary

active partner

The *communication partner* that sets up a *connection* to another *TS application*.

API (application program interface)

APIs are program interfaces that provide the functions of a program system. As the programmer, you use the APIs when programming applications. APIs offer functions for connection management, data exchange, and mapping names to addresses. APIs in the CMX environment are sockets, ICMX, XTI, and TLI.

application

An application is a system of programs which implements a particular range of services of a DP system in order to provide a higher-quality service to the human or electronic user. Communication applications are applications that use the communication functions of a DP system in order to provide global services when a network is in operation.

Most applications are qualified by a prefix which identifies the underlying service range (CMX application, UTM application, DCAM application, Motif application, Windows application, etc.). Examples of communication applications are file transfer, terminal emulation, electronic mail, world wide web browser and server, transaction systems such as UTM, and in general all applications based on the client-server principle.

CC (communications controller)

A CC is a component for connecting a UNIX system to a network. You need a CC to physically attach your system to a subnetwork, unless the interface is integrated on a different module, e.g. the motherboard (onboard interface).

To obtain a logical connection to the network, CCs are generally operated with an associated *communication control program* (*CCP*). These CCs are known as loadable CCs. EWAN, CCA, CCS0, LCEII, PWS0[_U], PWS2[_U] and PWXV[_U] are examples of loadable CCs for connecting to X.25 and telephone networks, ISDN and Ethernet. Loadable CCs are generally controlled by a *subnetwork profile*. The subnetwork profile is a component of the *CCP*.

CCP (communication control program)

A CCP is a program system (software product) which, together with one or more *CCs*, provides the logical access of a UNIX system to a *network*. A CCP implements the four lower layers (transport system) of the OSI Reference Model for data communication. CCP-WAN, CCP-ISDN, CCP-ETHN, and CCP-FDDI are examples of CCPs for connecting to X.25 and telephone networks, ISDN, Ethernet, and FDDI.

A CCP comprises a number of components, the *subnetwork profile* and *transport service providers*.

communication partner

A *TS application* that maintains a virtual connection to another *TS application* and exchanges data with it.

connection establishment

The phase in connection mode that enables two transport users to create a transport connection between them.

connection release

The phase in connection mode that terminates a previously established transport connection between two users.

GLOBAL NAME of an application

Each *CMX application* identifies itself and its communication partners in the network by symbolic, hierarchical GLOBAL NAMES. A GLOBAL NAME consists of up to five name parts (NP[1- 5]), which you can use to define the application (NP5), the processor (NP4), and (up to three) administrative domains (NP[3-1]).

Example: The GLOBAL NAME "YourApplication.D018S065.mch-p.sni.de" means: "YourApplication" resides on the host "D018S065" in the domain "mch-p.sni.de".

When you, as administrator, are choosing a GLOBAL NAME, you must adhere to the regulations and recommendations of the specific application.

As the administrator, you can use the graphical user interface *CMXGUI* to assign a *TRANSPORT ADDRESS* or a *LOCAL NAME* of an application to the GLOBAL NAME of the application on a 1:1 basis. As the programmer, you can obtain the TRANSPORT ADDRESS or LOCAL NAME expected by CMX from the GLOBAL NAME using the function calls of the *transport name service* (TNS).

LOCAL NAME of an application

A CMX application uses the LOCAL NAME to attach to CMX in its local system for communication. The LOCAL NAME comprises one or more *T-selectors*, which identify the transport system via which the CMX application is to communicate. As the administrator, you can enable or disable the communication of a CMX application via particular transport systems and fulfill any requirements of the CMX application for specific T-selector values, e.g. in file transfer.

Example: An application is to use the T-selector “cmxappl” (in lowercase letters!) for communication via the TCP/IP- RFC1006 transport system, and the T-selector “$CMXAPPL” (in uppercase letters!) for communication via the NEA transport system.

As the administrator in CMX, you can use the graphical user interface *CMXGUI* to assign the LOCAL NAME of an application to the *GLOBAL NAME* of the application. As the programmer, you can obtain the LOCAL NAME expected by CMX from the GLOBAL NAME using the function calls of the *transport name service* (TNS).

OSI Reference Model

Open Systems Interconnection is the communication architecture defined by the International Organization for Standardization (ISO) in ISO standard 7498. This architecture defines reliable data interchange between applications running on different hardware platforms. To perform this complex task, the OSI Reference Model distinguishes between seven interoperating subtasks, each of which is implemented on a particular layer. The lower four layers represent the *transport system*, while the top three layers represent the view of the *application*, e.g. the data formats.

partner

see *communication partner*.

passive partner

The *communication partner* that does not set up a *connection* itself but is addressed by another communication partner.

process

A process is a program during execution. It consists of the executable program, the program data, and process-specific administration data required to control the program.

property

Attribute of a *TS application* in the *TS directory*, where the application is registered together with the *GLOBAL NAME*.

TEP

XTI transport endpoints and *TS application* processes attached to CMX.

TNS (transport name service)

The TNS is a component of *CMX* which supports the correct mapping of the *GLOBAL NAMES* of *CMX applications* in the network to *TRANSPORT ADDRESSES* and *LOCAL NAMES*. As the administrator, you configure your chosen assignment of GLOBAL NAME to TRANSPORT ADDRESS for remote applications, as well as the assignment of GLOBAL NAME to LOCAL NAME for local applications. As the applications programmer, you can use these maps via an *API* and thereby work solely with the GLOBAL NAMES of applications without assessing the maps.

The TNS provides network-wide identification of applications by means of logical GLOBAL NAMES and their mapping to corresponding *network addresses*. This means that you can identify applications without having to know their network addresses. Together with the *FSS*, the TNS provides a complete mapping of the logical name to a concrete *subnetwork address* and a *route* through the various subnetworks of the network.

TRANSPORT ADDRESS of an application

A calling *CMX application* transfers the TRANSPORT ADDRESS of a called communication partner to *CMX* when communication is being established. CMX uses the TRANSPORT ADDRESS to locate the communication partner in the network and determine a *route* through the network. The TRANSPORT ADDRESS generally depends on the logical and physical structure of the network (and its subnetworks). The TRANSPORT ADDRESS contains the specifications of your network operator(s) which are specific to your network. As the administrator, you can influence the TRANSPORT ADDRESS and hence the communication paths independently of the application.

The components of a TRANSPORT ADDRESS are: a network address for uniquely identifying the remote system on which the application resides, the type of *transport system* via which the remote application can be reached, and the *T-selector* that identifies the remote application in the remote system.

Examples of network addresses are: the Internet address in dot notation "192.11.44.1", the NEA network address in the notation processor/region number "47/11", and the X.25 address (DTE address) as a string of digits "45890010123".

As the administrator, you can use the graphical user interface *CMXGUI* to assign a TRANSPORT ADDRESS of the application to the *GLOBAL NAME* of the application on a 1:1 basis. As the programmer, you can obtain the TRANSPORT ADDRESS expected by CMX from the GLOBAL NAME using the function calls of the *transport name service* (TNS).

transport connection
The communication circuit that is established between two transport users in connection mode.

transport layer
Fourth layer in the *OSI Reference Model*; described in ISO standard 8072.

transport reference
A number which uniquely identifies a *connection* within a *TS application*.

transport system
The transport system is represented by the four lower layers of the *OSI Reference Model*. A *CCP* implements the four layers of the transport system. The transport system guarantees the secure exchange of data between systems whose *applications* communicate with each other, regardless of the underlying network structures. The transport system uses protocols for this purpose.

transport service access point (TSAP)
A TSAP is a uniquely identified instance of the transport provider. A TSAP is used to identify a transport user on a certain end system. In connection mode, a single TSAP may have more than one connection established to one or more remote TSAPs; each individual connection then is identified by a transport endpoint at each end.

TS application
Transport service application:

A TS application is an application that uses the services of the transport system. It consists of programs that can set up a virtual *connection* to another TS application in order to exchange data with it.

TS directory

Database containing information about *TS applications*. The TS directory is managed using the *Transport Name Service* in UNIX.

T-selector

The T-selector identifies a communication application within the system on which the application is running. Together with the *network address* of the system, the T-selector forms the *TRANSPORT ADDRESS* of an application which uniquely identifies this application within the network. The format and value range of the T-selector depend on the type of *network*. In the NEA network, the T-selector corresponds to the station name (e.g. T'DSS01').

TSP (transport service provider)

A TSP is a component of a *CCP* or of *CMX* which, with the exception of the NTP (null transport), provides the OSI transport service in the network using a transport protocol. As the administrator, you can determine the usage of a particular TSP for the communication of *applications*. RFC1006 is the TSP in CMX which, together with TCP/IP, provides the OSI transport service in the Internet. NTP (null transport) offers *CMX applications* direct access to the network services of the X.25 subnetwork. TP0/2, TP4, and NEA are the TSPs for an OSI environment and the TRANSDATA network.

Together with a *subnetwork profile*, a TSP forms a *transport system*. It offers a set of configurable runtime and tuning parameters, assesses the *TRANSPORT ADDRESS*, and finds a suitable route through the network. To do this, the TSP uses your specifications in the *FSS*, if necessary.

Abbreviations

CC
Communication Controller

CCP
Communication Control Program

CMX
Communication Manager in UNIX

ETSDU
Expedited Transport Service Data Unit

IP
Internet Protocol

ISO
International Organization for Standardization

LAN
Local Area Network

NEA
Network architecture in TRANSDATA systems

NLS
X/Open Native Language System

OSI
Open Systems Interconnection

PDN
Program system for teleprocessing and network control

TCP
Transmission Control Protocol

TNS
Transport Name Service in UNIX

TS
Transport Service

TSAP
Transport Service Access Point

TSDU
Transport Service Data Unit

UDP
User Datagram Protocol

WAN
Wide Area Network

XTI
X/Open Transport Interface

Tables

Table 1: General XTI calls 3
Table 2: Functions for accessing the Name Service 5
Table 3: Header files and applications 5
Table 4: ISO transport services 37
Table 5: NEA transport service 37
Table 6: Message-oriented transport services 38
Table 7: Functions implemented in XTI V5.1 51
Table 8: Association of transport services with CCP profiles 69
Table 9: Values of the members in info 71
Table 10: Reasons for connection release 76
Table 11: Reasons for connection release (XTI) 78
Table 12: Relationships between XTI events and poll() bits 96
Table 13: Event bits to be set for applications running also on non-UNIX systems 96
Table 14: Event class and event bits at select() 97
Table 15: Event class and event bits at select() on non-UNIX systems . 97

References

[1] **CMX V5.1** (Solaris)
Communications Manager UNIX
Operation and Administration
User Guide

Target group
System administrators and users

Contents
The manual describes the function of CMX as mediator between applications and the transport system. It contains basic information on configuration and administration of systems in network environments.

[2] **CMX V5.1**
Communications Manager UNIX
Programming Applications
Programmers Reference Guide

Target group
Programmers

Contents
The manual describes the program interface of CMX, i.e. all tools that you can use for developing TS applications.

[3] **Reliant UNIX 5.45**
Network Programming Interfaces
Programmer's Guide

Target group
Application programmers

Contents
This manual describes how to write application programs that use the Reliant UNIX networking facilities such as TLI (Transport Layer Interface), Sockets and RPC (Remote Procedure Call).

[4] **Solaris 7 Reference Manual Collection**
man-Pages (3), Library Routines

[5] **X/Open Networking Services (XNS), Issue 5.2**
ISBN 1-85912-241-8

Ordering manuals

Please apply to your local office for ordering manuals.

Index

A
address
    TS application 9
address management 9
addressing 9

C
CCP
    assignment of transport service 69
collective service 69
communication software
    architecture 6
concatenate expedited data 83
connection 80
connection establishment 108
connection release 108

D
data unit 81, 87
DIR.X 39

E
environment variable
    LANG 61
error messages
    format of 61

F
function calls
    supplements 51
functions
    implemented 51

G
GLOBAL NAME 10
    example 13
    map 30
    structure 11
global name
    get 26

H
header file 5

I
implemented functions 51
installation 7

K
kernel components 7

L
LANG
    environment variable 61
LeafEntities 11
leaves 11
library trace 102
    controlling/activating 101
    edit 104
    output 105
library trace program 6
LOCAL NAME 14
    structure 15
local name
    get 23
LOCAL NAMES
    transport service 16

M
map transport address 32
message texts
    format of 61

N
name
    ascertain 19
    transport service 67
    TS application 9
name part
    GLOBAL NAME 11
Name Service 5

name structure
    GLOBAL NAME 11
naming tree
    GLOBAL NAME 11
netconfig 37
netdir_getbyaddr() 32
netdir_getbyname() 30
netdir_options() 36
network address 15
network configuration file 37
NLS 61
nodes 11
NonLeafEntities 11
notational conventions 101

P
poll 96
properties 10
    TS application 14
property
    ascertain 19

R
read from a file 99
read() 99
reasons for connection release
    list of 76
receipt 80
release 86
ROOT 11
root 11
ROUTING INFORMATION 15
runtime environment
    XTI application 7

S
select 97
service access point 15

T
t_accept 54
t_alloc 56
t_bind 57
t_connect 59
t_error 61
t_free 62
t_getaddr 20
t_getinfo 63
t_getloc 23
t_getname 26
t_listen 64
t_look 66
T_MORE flag 83
t_msg 67
t_neat 67
t_open 67
t_optmgmt 91
t_osi_cots 68
t_rcv 73
t_rcvconnect 74
t_rcvdis 76
t_rcvrel 80
t_rcvudata 81
t_rcvuderr 82
t_rfc1006 68
t_sinix 68
t_snd 83
t_snddis 85
t_sndrel 86
t_sndudata 87
t_sync 88
t_sysconf () 89
t_tp2 68
t_tp4 68
t_unbind 90
taddr2uaddr() 34
TRANSPORT ADDRESS 15
    map 34
    structure 15
transport address
    get 20
transport connection 111
transport service
    assignment of CCPs 69
    collective service 69
    LOCAL NAMES 16
    name of 67
transport service access point 15

transport service provider 6
TRANSPORT SYSTEM 15
TS application
  names and addresses 9
  properties 14
TS directory 9, 10
TSAP 15
T-selector 15
  transport service 16

U
uaddr2taddr() 35
unitdata 81
universal address
  map 35

X
X/Open Native Language System 61
XTI
  installation 7
XTI application 3
XTI calls
  supplements 51
xtil 104
  output 105
XTITRACE 101, 102