1 Preface

1.1 Brief product description of CMX(BS2000)

The transport access system CMX(BS2000) is one of the BS2000 products (DCAM, CMX(BS2000), SOCKETS(BS2000)) that provide an interface to the BCAM (**B**asic **C**ommunication **A**ccess **M**ethod) transport system. This interface is also available in the SINIX and MS-DOS operating systems. CMX(BS2000) can be used to create application programs that can communicate with other applications irrespective of the transport system.

1.2 Target group

This manual is intended for programmers who develop transport service (TS) applications. These TS applications are used for communication, and consist of application programs implemented in C.

In order to work with CMX(BS2000), you must be familiar with the C programming language and the C development system. Knowledge of the principles and methods of data communications will also prove helpful, in particular of the OSI Reference Model as standardized in ISO 7498.

1.3 Summary of contents

This manual describes the CMX(BS2000) program interfaces, i.e. all the tools you will need in order to develop TS applications of your own.
Diagnostic information is included in the appendix.

Structure of the manual

The manual is divided into two parts:

Part 1 is intended to help you get acquainted with CMX and focuses on helping the first-time user to create TS applications.
This part describes the mapping of a TS application onto the task concept of your system and the allocation of transport connections to tasks of the TS application. The structure of a TS application is explained, showing how it can be divided into three communication phases and how the functions of the program interfaces are used within these phases. In addition, you will learn how to obtain diagnostic information from CMX(BS2000) in the case of errors. To explain the individual programming steps, program fragments have been provided as examples.

Part 2 consists of the chapter entitled "The ICMX program interface". This chapter gives a detailed description of the CMX(BS2000) program interface and each individual function call of this interface and its parameters. The description is arranged in alphabetical order. The chapter begins with a summary of all the information you will need to use the functions.

The description takes into account all the various ways of connecting a computer to a network (LAN and WAN).

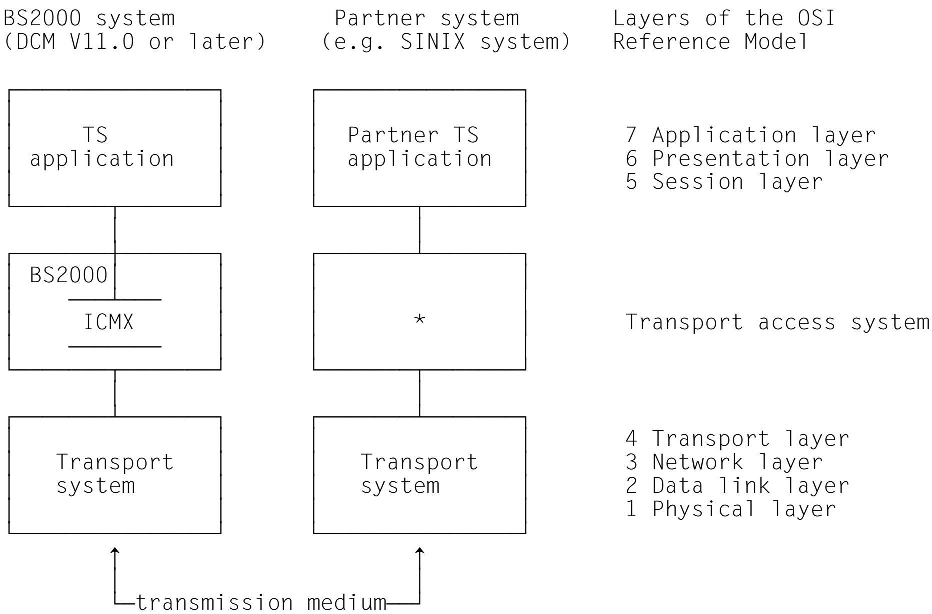
The program interface is described independent of which operating system is used.

2 The CMX(BS2000) transport access system

Any application that wishes to exchange data with another application in some other end system requires the services of a transport system. The transport system performs all the necessary tasks to set up the connection and to transport data over the physical media (lines, computers). Applications that use the services of a transport system are called TS applications.

A TS application should be capable of setting up connections and exchanging data using different connection mechanisms. As far as possible, the TS application should be independent of the underlying connection mechanism. These may differ in various respects, e.g. the size of the data unit that can be transferred, the format of the partner application's transport address to be passed, and the format of the TS application's address in the local system. For this reason, CMX(BS2000) provides TS applications with a uniform interface known as the ICMX program interface. This interface provides TS applications with access to the services of transport systems that conform to the standards laid down in the OSI Reference Model. CMX(BS2000) is thus a transport access system.

2.1 Communication between TS applications



* = Transport access system in the partner system

The CMX(BS2000) transport access system

A TS application that uses CMX(BS2000) functions can thus communicate in a uniform way with the following TS applications:

– other TS applications in the same computer (local communication),

– TS applications in other SINIX or SINIX-ODT computers which use the functions of the CMX(L) transport access system,

– TS applications in host computers running BS2000 and using the functions of the DCAM and CMX(BS2000) transport access systems, or of UTM.

– TS applications in communication computers running PDN and using the functions of the CAM transport access system,

– TS applications in systems of other vendors, assuming they conform to the standards described in the OSI model (for example, ICP/IC according to RFC 1006)

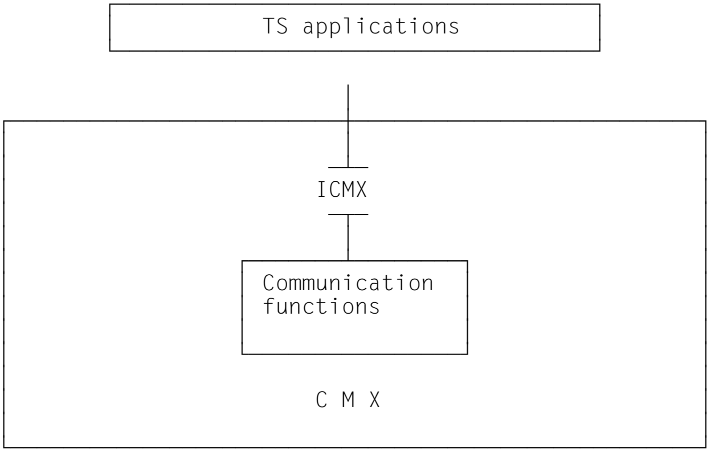
For the programmer, the ICMX uniform program interface means that he or she can develop TS applications independent of specific data transmission characteristics, i.e. only the ICMX functions need to be programmed for communication. These functions can be used to:

- attach the TS application to CMX(BS2000),
- set up transport connections to partner applications,
- send and receive data,
- control the data flow,
- disconnect transport connections,
- detach TS applications from CMX(BS2000).

TS applications that use the functions of CMX(BS2000) interfaces are also called CMX(BS2000) applications in the description below. This term is always used whenever it is necessary to make a distinction between TS applications running under CMX(BS2000) and other TS applications.

2.2 The CMX(BS2000) program interfaces - an overview

CMX(BS2000) provides the programmer of TS applications with functions for connection-oriented communication. These functions cover local services, connection handling and data exchange. They are available via the ICMX program interface.



CMX(BS2000) program interface

The CMX(BS2000) program interface is a library interface, i.e. the functions of CMX(BS2000) are provided in the form of a connection module (YDCMXLNK) and a large library module (YDCMXLIB). The connection module is located with the *cmx.h* header file in the SYSLIB.CMX.010 library. The large library module is provided in the SYSLNK.CMX.010 library and is installed as the CMX-TU subsystem.

2.2.1 CMX(BS2000) functions for communication (ICMX)

The CMX(BS2000)(ICMX) program interface includes all functions which are used by a TS application for communication.

The following function groups are provided at the program interface:

Functions for attaching to and detaching from CMX(BS2000)

When a TS application attaches itself to CMX(BS2000), it passes its LOCAL NAME, i.e. its own address within the local system, to CMX(BS2000). Only then is the TS application addressable. After communication, the TS application must detach itself from CMX(BS2000).

Functions to establish a connection

This includes the following functions:

– active connection setup:
  The two functions in this group are used to request a connection with a remote TS application (connection request), and to set up the connection after receipt of a positive response from the remote TS application (connection confirmation).

– passive connection:
  The two functions in this group serve to accept a connection setup request from a remote TS application (connection indication) and to respond to this request (connection response).

Functions to close down a connection

The two functions in this group are used to actively close down a connection (disconnection request), or to accept a disconnection request (disconnection indication).

Functions to redirect a connection

Within a TS application, a connection may be passed on (redirected) to another task of the same TS application. The two functions in this group can be used to redirect a connection and to accept a connection (redirect request) from another task (redirect indication).

Functions for the exchange of data

These functions allow you to exchange data as follows:

- send (data request) and receive (data indication) normal data.
- send (expedited data request) and receive (expedited data indication) expedited data.

  Expedited data refers to small amounts of data that can be transmitted to a communication partner with priority over the main data stream. These functions are optional.

Functions for flow control

If you currently cannot or do not wish to receive any data, you can have the data flow stopped by informing CMX(BS2000). CMX(BS2000) will then stop signaling incoming data. The communication partner is (usually) notified, and will not be permitted to send you any further data until you release the data flow. The data flow can be controlled separately for normal and expedited data (datastop, datago, xdatstop, xdatgo).

Functions to request information

This group includes functions that can be used to:

- await or fetch an event (event).
  A typical example of an event is a disconnection request from the communication partner.
- request information on errors (error).
- request information on CMX(BS2000) parameters (info).
- query LOCAL and GLOBAL NAMES, and TRANSPORT ADDRESSES (get local name, get name, get address).

Function for synchronizing other events

This function can be user for wakening a task (another or its own) from the waiting status (*wake*).

The use of the functions in the programs of a TS application is explained in the chapters entitled "Event processing and error handling, "Attaching to/detaching from CMX(BS2000)", "Managing connections" and "Transmitting data". The function calls are described in detail in the chapter entitled "The ICMX program interface"

2.2.2 System and user options

The functions of the CMX(BS2000) program interfaces consist of mandatory and optional functions with mandatory and optional parameters.
For communication with partners via CMX(BS2000), the mandatory functions with the mandatory parameters are always available for all transport connections. Depending on the type of network interface, optional functions are also available, as well as optional parameters for the mandatory functions.

The options are the following:

Option	**optional function**	**optional parameter**	**s/u**
User data at connection setup	n	y	s/u
User data at disconnection	n	y	s/u
Expedited data	y	y	s/u
Monitoring of inactive time *)	n	y	s/u
Connection limit, active/passive mode *)	n	y	u
User reference of attachment	n	y	u
User reference of connection	n	y	u
Time limit on synchronous event processing (?60 sec.)	n	y	u
Waiting time for connection redirection *)	n	y	u
n/y = no/yes * = not in CMX(BS2000) s = System option u = User option			

Table 1: CMX(BS2000) options

The system options are oriented to the functionality of the transport connections used. If options are used that the transport system or the communication interface of the partner application does not provide, the connection will not be established, or a disconnect indication will be issued by CMX(BS2000). Given an appropriate transport system, CMX(BS2000) guarantees error-free execution of your CMX(BS2000) application.

If communication is to be error-free, the user options must also be correct, i.e. the partners must have a common understanding of how they are used.

This means that CMX(BS2000) does *not* compensate for the difference between the functionality expected in the TS application and that actually provided by the transport system. This applies particularly to the system options shown above.

3 TS applications

This chapter outlines the characteristics of TS applications that use the functions of the CMX(BS2000) program interfaces.

The following points are covered in the sections of this chapter:

– Name and properties of a TS application
  Every TS application has a GLOBAL NAME, with which it can be uniquely identified within the network. To communicate with other TS applications in the network, a TS application must be addressable. For this reason, a TS application is assigned the properties TRANSPORT ADDRESS and LOCAL NAME in addition to other properties.

– Structure of a TS application
  A TS application is a C program or a system of C programs that calls CMX(BS2000) functions.
  This section describes what is required when writing TS application programs, how such C programs are compiled, and which libraries must be linked into the source code.

– Association between a TS application, tasks, and connections
  This section deals with the question of how a TS application can be mapped onto a system's task concept, and illustrates the association between a task and a connection.

3.1 Names and addresses of TS applications

Every TS application has a GLOBAL NAME. This name identifies the TS application uniquely in the network, i.e. different TS applications have different GLOBAL NAMES. The GLOBAL NAME specifies which TS application is involved.

The GLOBAL NAMES of all TS applications in the local system and those of all TS applications in remote systems with which the local TS applications wish to communicate are recorded in a name and address directory of the local transport system.

In BS2000, the name and address directory is implemented by the BCAM mapping administration function (see /BCMAP command in the manual "BS2000 System Operator's Guide). The generation of the entries in the Transport Name Service is the responsibility of the system or network administrator, and is therefore not dealt with in this manual. In order to generate the entries, the developer of a TS application must inform the administrator of the names of his/her own TS applicatio, the names of all accessible partners, and the type of connection.

3.1.1 The GLOBAL NAME of a TS application

The GLOBAL NAME of a TS application is a hierarchically structured name consisting of up to 5 name parts: name part[1] through name part[5]. Of these, name part[1] is the highest in the hierarchy and name part[5] the lowest. All levels of the hierarchy need not be present in a GLOBAL NAME; it is possible to omit name parts. A GLOBAL NAME can also consist of a single name part at any hierarchy level.

Examples of GLOBAL NAMES are given below:

	Np[1]	**Np[2]**	**Np[3]**	**Np[4]**	**Np[5]**
GLOBAL NAME 1	D	Siemens AG	Mch-P	DF1	G._Meier
GLOBAL NAME 2		Dept A	Reg18	Proc. 1	$DIALOG
GLOBAL NAME 3	49	089	636		47658

Np = Name part

The GLOBAL NAMES are written in the C procedures as in SINIX (name parts separated by a period ".", for example *franz.xyz.1*). When entering them in the Transport Name Service (*/BCMAP* command), the name parts must be separated by X'00'.

Example:

```
#include        <stdio.h>
#include        <cmx.h>
     .
     .
struct t_myname *p_myname;
     .
     .
if ((p_myname = t_myname("franz.xyz.1",NULL)) !=NULL
{
     .
     .
}
else t_perror("error in t_getloc",t_error());
```

The name "franz.xyz.1" (=X’86998195A94BA7A8A94BF1’) is allocated the T-selector "TEST001" for the LOCAL NAME using the following BCAM command:

```
/BCMAP FU=DEF,SUBFU=LOCAL,APPL=(OSI,X'86998195A900A7A8A900F1'),
TSEL-I=(8,C'TEST001')
```

(see also the chapter entitled "Program examples").

3.1.2 The properties LOCAL NAME and TRANSPORT ADDRESS

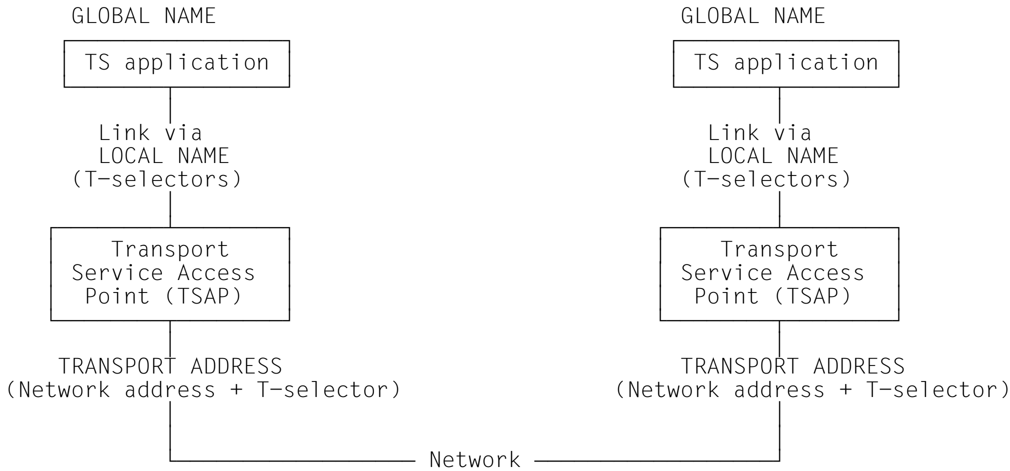
Every TS application is assigned a unique Transport Service Access Point (TSAP) when it is attached to CMX(BS2000). The TSAP is identified by means of the LOCAL NAME that is specified by the TS application when it attaches itself to CMX(BS2000).

The TS application can access the services of the transport system via the TSAP. Which transport systems, i.e. network interfaces, can be accessed by the TS application will depend on the T-selectors contained in the LOCAL NAME of the TS application. The LOCAL NAME contains one or more T-selectors. A single T-selector can be valid for multiple network interfaces, provided these are of the same type.

The TS application can be addressed from the network via the T-selector, since the T-selector is a component of its TRANSPORT ADDRESS for the respective network. The TRANSPORT ADDRESS provides a means of iniquely addressing the TS application in the entire network. The TRANSPORT ADDRESS of a TS application consists of the network address of the end system in which the TS application is located, and the T-selector of the TS application for this network unterface. The TRANSPORT ADDRESS is thus structured as follows:

TRANSPORT ADDRESS =
    end system network address + (locally unique) T-selector

The following diagram illustrates the relationship between the LOCAL NAME, TSAP, and TRANSPORT ADDRESS.



TRANSPORT ADRESS and LOCAL NAME

The *t_getloc()*, *t_getaddr()* and *t_getname()* calls provided at the CMX(BS2000) program interface can be used to determine the LOCAL NAME or TRANSPORT ADDRESS for a given GLOBAL NAME, and the GLOBAL NAME corresponding to a given TRANSPORT ADDRESS. The TS application must carry out name-address conversion with these calls. The contents of the address structures generated may not be evaluated or changed by the TS application. This ensures that the application will not be affected by possible changes in the address structure.

3.2 Structure of a TS application

A TS application is a C program or a system of C programs that call CMX(BS2000) functions. This chapter describes what should be observed when creating such a program.

The following figure illustrates the structure of a program of this type. The specified function calls are part of the ICMX interface.

```
#include <cmx.h>
 .
 .
main(argc, argv)
int argc;
char *argv[];
{
        .
        .
        /* 1st communication phase */

        t_getloc();             /* Ascertain LOCAL NAME */
        t_attach();             /* Attach to CMX(BS2000) */

        /* 2nd communication phase */

        t_getaddr();            /* Ascertain TRANSPORT ADDRESS */
                                /* of partner */
        t_conrq();              /* Set up connection */
        .
        .
        t_concf();              /* Accept connection confirmation */

        /* 3rd communication phase */

        t_datarq();             /* Send data to partner */
        .
        .
        t_datain();             /* Receive data from partner */
        .
        .
        t_disrq();              /* Close down connection */
        t_detach();             /* Detach from CMX(BS2000) */
        .
        .
        exit();
}
```

Structure of a TS application program in ICMX

Header file

Every TS application program must contain an include statement for the file <*cmx.h*>. <*cmx.h*> contains the definitions of the parameters for the functions of the ICMX interface. This header file is located in the *SYSLIB.CMX.010* library.

Permissible order for calling CMX(BS2000) functions

TS application programs must call CMX(BS2000) communication functions in a certain order. The process of communication can be divided into three phases. A TS application must pass through each phase successfully before it can enter the next phase.

1st communication phase:
The TS application must attach itself to CMX(BS2000). Only when the TS application is known to CMX(BS2000) can it make use of the services of CMX(BS2000). The operations of this communication phase are described in the chapter entitled "Attaching to/detaching from CMX(BS2000)".

2nd communication phase:
In this phase the TS application sets up the connection to its communication partner. During connection setup the two partners must reach an agreement as to how the subsequent exchange of data is to take place and what form the data is to have. Both partners determine, for example, whether they wish to exchange expedited data. The operations of this communication phase are described in the chapter entitled "Managing connections".

3rd communication phase:
In the third phase the data is exchanged between the partners. Both communication partners can send and receive data. The operations of this communication phase are described in the chapter entitled "Transmitting data".

This is the order in which a TS application program may call CMX(BS2000) functions. In addition, note that some calls may be issued only after certain responses from the other communication partner have arrived and been received by the TS application (see the section on "Event processing"). One might say that a TS application assumes various states during the course of communication. Several states are possible within each communication phase. Only certain transitions are possible between the states within a given phase and between states of different phases.

A TS application can shift from one state to the next only by calling certain CMX(BS2000) functions or when certain events arrive for it from the network. These are represented in diagram form in the section entitled "States of TS applications and state transitions". These diagrams should make it easier to create TS application programs.

Reaching an agreement as to the form of transferred data

Two TS applications wishing to communicate with each other must also reach an agreement as to the form of the data to be transferred. The TS applications themselves must carry out the necessary recoding, as data transfer through the transport system and CMX(BS2000) is code-transparent. Of importance here is the character set in use in each system. In SINIX and SINIX-ODT computers, this is the ISO 7-bit code; in BS2000 and PDN systems, it is the EBCDIC code.

Parameter passing and storage allocation

In TS applications parameters are passed to CMX(BS2000) functions as values or pointers; for options, unions (union ...) are defined. All structures are declared in the header files. In your program, you must always provide all storage areas used to pass values to CMX(BS2000), or in which CMX(BS2000) is to return anything. You allocate such storage areas either at compile time (statically) or at runtime (dynamically), e.g. with *malloc()* (see the description of the C library functions (BS2000)). In the CMX(BS2000) parameter structures, length fields are defined for areas of variable length. Before calling CMX(BS2000), enter in these fields the lengths of the areas provided. Then, upon return, you can usually read from these fields the lengths of the data returned by CMX(BS2000).

3.3 Compiling and linking TS application programs

After a C program *prog.c* of a TS application has been edited, is must be compiled using the Siemens C compiler (V1.0 or later); the CMX(BS2000) functions from the CMX(BS2000) library *SYSLIB.CMX.010* must then be linked into the program. The C-RTS must also be linked in, as must the *YDCMXLNK* module from the *SYSLIB.CMX.010* library. This module connects to the CMX-TU subsystem, which implements the actual CMX library.

The advantages of this are as follows:

– The linked programs are considerably smaller.

– The application programs need not be relinked after a library has been switched (possibly for maintenance purposes).

Please refer to the system manuals or the Release Notice for further information on the subsystems.

Note

CMX stores task-specific data in the *YDCMXLNK* module. For this reason, the module may not be write-protected, and may not be linked to SHARED CODE modules.

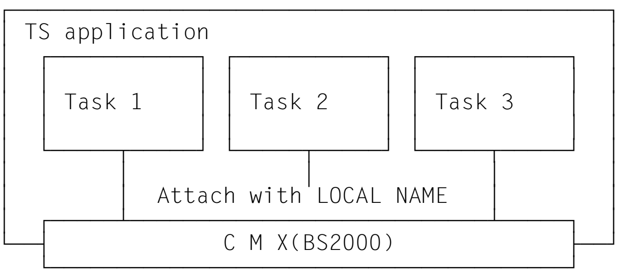
3.4 TS applications, tasks, connections

The two following sections describe the relationships of TS applications to tasks and of tasks to connections.

3.4.1 TS applications and tasks

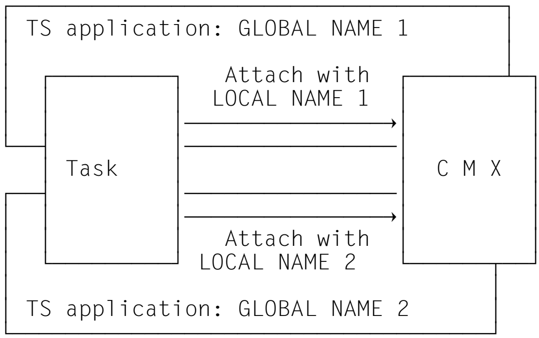
In the simplest case a TS application is implemented in a single task. However, there are additional possibilities for structuring a TS application.

A TS application can work with multiple tasks, which need not be started under the same user ID, but can run under various IDs. Each task of a TS application must attach itself to CMX(BS2000) separately. Tasks belong to the same TS application when they have attached themselves to CMX(BS2000) using the same LOCAL NAME. The first task to attach itself creates the TS application.



One TS application - multiple tasks

On the other hand, one task may control multiple TS applications. To achieve this, you attach the task to CMX(BS2000) using different LOCAL NAMES.



One task - multiple TS applications

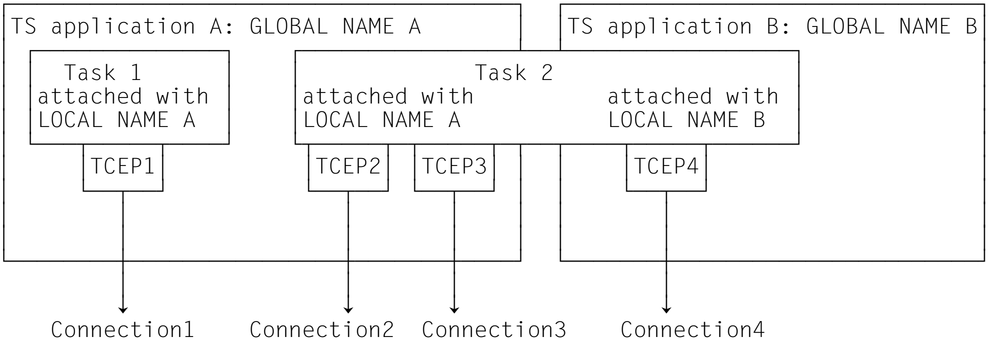
The task distinguishes the various TS applications it controls by means of the different LOCAL NAMES, or by means of a freely chosen user reference.

Like in SINIX, the CMX(BS2000) concept does not allow for asynchronous routines (contingencies).
If contingencies are used in spite of this, please note that only one CMX call at a time can be processed in the library. However, this does not apply to the *t_wake* call.

3.4.2 Connections and tasks

The tasks of a TS application can set up connections to other TS applications independently of one another, and individual tasks of the TS application may maintain multiple connections simultaneously. If the task is attached to more than one TS application, the connections may also belong to different TS applications. When the connection is set up, a Transport Connection Endpoint (TCEP) is created for each connection. In other words, a single task can serve a number of TCEPs, but the same TCEP may not be simultaneously assigned to multiple tasks. Each connection is assigned to **one and only one** one task at a given time.

CMX(BS2000) assigns each connection an identifier, known as the transport reference. This alone enables the task to address a specific connection.



Connections and tasks

A task may, however, redirect a connection to another task that has attached itself in the same TS application. The connection will then no longer be recognized in the task that redirected it. In this way, it is possible to handle connections to various partners in various tasks. A central distribution task may, for instance, receive all connections and then redirect them to appropriate subordinate tasks. In the above diagram for example, task 2 could redirect connection 2 or connection 3 to task 1.

4 Event processing and error handling

This chapter describes event processing and error handling for TS applications using CMX(BS2000).

4.1 Event processing

The operations involved during communication between TS applications are asynchronous, i.e. a wide variety of events can occur independently of the activity of TS application. Events are requests and responses received by CMX(BS2000) from other TS applications in the network, or messages from the transport systems involved.

Examples of such events are:

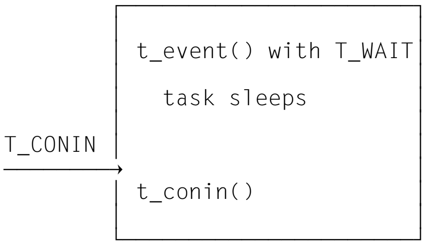
- The connection request of a communication partner (the "calling application")
- The arrival of data via an existing connection
- Flow control events (set and released send locks)
- Disconnection by the communication partner or CMX(BS2000)

CMX(BS2000) forwards these events to the TS application when the *t_event()* function is called by the TS application. Exactly one event is passed by CMX(BS2000) for each *t_event()* call, possibly with the identification of the connection involved (transport reference). The TS application must then directly process the received event as required, e.g. by calling the corresponding "fetch" function.

The CMX(BS2000) functions are designed in a manner that allows, but does not compel the TS application to wait for a possible answer from the network after issuing a call. There are two ways in which a TS application can process events.

1. Synchronous processing

The TS application calls *t_event()* with the parameter *cmode* = T_WAIT. As long as no event is waiting, the task sleeps and consumes no CPU time. When there is an event (T_CONIN in diagram below), CMX(BS2000) awakens the task, and *t_event()* returns the code of the event and, when appropriate, the transport reference of the connection involved.



Synchronous processing

The task can be awakened with *t_wake()*, even if it is sleeping in *t_event*. CMX(BS2000) then resumes it with T_NOEVENT.
When *t_event()* is called, it is also possible to limit the waiting time. Simply specify how long the task is to wait for an event. If no event arrives within this time, CMX(BS2000) will resume the task with T_NOEVENT.

2. Asynchronous processing

Call *t_event()* with the parameter cmode = T_CHECK. If no event is waiting, the call will immediately return with T_NOEVENT. You may continue with any processing and subsequently call *t_event()* again to check for a possible event.

However, it is not wise to just have *t_event()* run in a continuous loop; it is better to use synchronous event processing (cmode = T_WAIT).

```
           ┌─────────────────────────┐
           │ t_event() with T_CHECK  │
T_NOEVENT  │                         │
──────────→│   .                     │
           │   . processing          │
           │   .                     │
           │ t_event() with T_CHECK  │
T_CONIN    │                         │
──────────→│                         │
           │ t_conin()               │
           └─────────────────────────┘
```

Asynchronous processing

CMX(BS2000) expects a particular reaction, depending on which event was reported. Since program execution is determined by what events occur, the program logic can be largely encapsulated in a switch construction, whose cases are the various events (as in the sample programs).

4.2 Error handling

A function call resulting in an error (t_ ...) always returns with a global error indicator (T_ERROR). A more precise value is obtained by calling the error checking function (t_error()).

The values returned by *t_error()* are in hexadecimal form, and are used for diagnostic purposes.

The appendix contains a description of the format of CMX(BS2000) error messages, a table with the CMX error values, tables showing the allocation of the CMX error values to the BCAM return codes, and the meaning of the individual return codes.

5 Attaching to/detaching from CMX(BS2000)

A TS application comes into existence as soon as a task attaches itself to CMX(BS2000) using the application's LOCAL NAME. Each further task wishing to operate within this TS application must also attach itself to CMX(BS2000) for this TS application, i.e. by using the same LOCAL NAME.

Before a task terminates it must detach all of its TS applications from CMX(BS2000). When the last task of a TS application has detached itself from CMX(BS2000), the TS application no longer exists for CMX(BS2000).

5.1 Attaching to CMX(BS2000)

A task attaches itself to CMX(BS2000) via the ICMX program interface using the *t_attach()* call.

When doing this, the task must pass the LOCAL NAME of the TS application for which it wishes to attach itself to CMX(BS2000). The task must determine the LOCAL NAME prior to attachment, i.e. before the *t_attach()* call. To do this, it calls the ICMX function *t_getloc()* and passes to *t_getloc()* a parameter with the GLOBAL NAME of the TS application for which it wishes to attach itself. *t_getloc()* returns a pointer to a structure in which the LOCAL NAME is stored. This pointer is passed as a parameter in *t_attach()*.
Thus, the *t_getloc()* call must precede the *t_attach()* call.

When the first task of a TS application attaches itself, a Transport Service Access Point (TSAP) is created for the TS application. The TSAP is the point at which the transport services is accessible. It is assigned the LOCAL NAME of the TS application.

Each task of a TS application can:

- actively set up connections for the TS application. The TS application can then assume the role of the "calling TS application" in the subsequent connection setup phase.
- wait passively on behalf of the TS application for connection requests from other TS applications in the network. The TS application can then assume the role of the "called TS application" during the course of communication.
- accept connections that another task of the same TS application wishes to pass to it (i.e. accept connection redirection). A task of the same TS application is a task that has attached itself to CMX(BS2000) using the same LOCAL NAME.

A task can also attach itself for several different TS applications. To do this, it calls *t_getloc()* and *t_attach()* for each of these TS applications.

CMX(BS2000) accepts connection requests from remote TS applications on behalf of a TS application as soon as a task of the TS application has attached itself to CMX(BS2000). Incoming connection requests are initially forwarded by CMX(BS2000) to the first task in the TS application to attach itself.

Only after successful attachment can a task call other CMX(BS2000) functions, i.e. issue other *t_...()* calls.

5.2 Detaching from CMX(BS2000)

Before a task terminates, it calls *t_detach()*. *t_detach()* dateches the task from CMX(BS2000) for that TS application. First, however, all TS connections maintained by the task must be closed down (see the chapter entitled "Managing connections"). If the task does not do this, CMX(BS2000) implicitly closes down all TS connections itself. This is provided only for exceptional situations, for example when a task is terminated prematurely.

When the last task of a TS application has detached itself, the TS application no longer exists for CMX(BS2000). Connection requests from remote TS applications will no longer be accepted for that TS application.

5.3 Examples of attaching and detaching a task

Example of attaching and detaching a task at ICMX

The following program fragment shows what happens when a task is attached and detached at the ICMX interface.
A task attaches itself to CMX(BS2000) for the TS application "Test-application-ACT" and then detaches itself. In the option structure *t_opta2*, it specifies that it only wishes to actively set up connections in this TS application (T_ACTIVE), and that no more than one connection is to be simultaneously maintained. However, this data is ignored by CMX(BS2000) Version 1.0, i.e. several connections can be set up both actively and passively.

```
#include        <stdio.h>
#include        <cmx.h>
.
.
#define ERROR   1
 .
 .
struct  t_opta2 t_opta2 = { T_OPTA3, 0, 0, };     /* t_attach () */
 .
 .
 /* Structures for addressing */

#define MYNAME  "Test_application_ACT"
char *myname = { MYNAME } ;
struct t_myname t_myname, *p_myname;
   .
   .
/* Attach active application to CMX(BS2000) */

if ((p_myname = t_getloc(myname, NULL)) != NULL)
        t_myname = *p_myname;
else {
       fprintf(stderr, ">>> ERROR 0x%x in t_getloc\n", t_error());
       exit(ERROR);
}
if (t_attach(&t_myname, &t_opta2) == T_ERROR) {
       fprintf(stderr, ">>> ERROR 0x%x in t_attach\n", t_error());
       exit(ERROR);
}
fprintf(stderr, "Application '%s' attached.\n", myname);
  .
  .
/* Detach TS application from CMX(BS2000) */
if (t_detach(&t_myname) == T_ERROR)
        fprintf(stderr, ">>> ERROR 0x%x in t_detach\n", t_error());
fprintf(stderr, "Application '%s' detached.\n", myname);
   .
   .
```

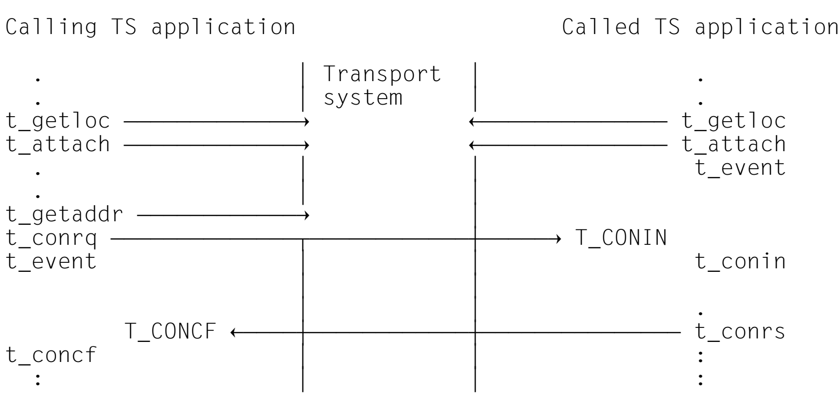
6 Managing connections

Connection setup and disconnection involve two TS applications. One is the calling TS application; it initiates connection setup. The other is the called TS application, with whom the calling TS application wishes to establish a connection. The following sections elucidate the relationships and sequences.

The fact that CMX(BS2000) is displayed only once in the diagrams is just a simplification of the presentation. Actually, each partner uses "its" CMX(BS2000) in its processor with the network and the transport systems in between.

6.1 Establishing a connection

The processing sequence in the course of setting up a connection at ICMX is explained first. The following figure illustrates the chronological sequence of ICMX calls in the programs of the calling and called TS application.



Establishing a connection (ICMX)

Connection setup in the calling TS application

The calling TS application first obtains its LOCAL NAME, and then attaches itself to CMX(BS2000). It then ascertains the TRANSPORT ADDRESS of the called TS application with *t_getaddr()* and requests a connection using *t_conrq()*. It then waits with *t_event()* for confirmation from the alled TS application, i.e. for the TS event T_CONCF. When *t_event()* has reported the TS event, the calling TS application establishes the connection with the call *t_concf()*.

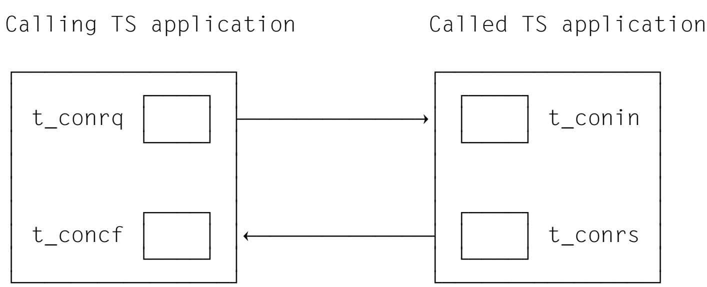
Connection setup in the called TS application

After being attached, the called TS application initially waits for a TS event with *t_event()*. The TS event T_CONIN indicates the connection request of the calling TS application. The called TS application accepts the connection request with *t_conin()* and answers it with *t_conrs()*.

Exchanging user data during connection setup

The reason the calls *t_conin()* (connect indication) and *t_concf()* (connect confirmation) are required is that both TS applications can already exchange user data while the connection is being set up, if the transport system supports this option (see section entitled "System and user options).

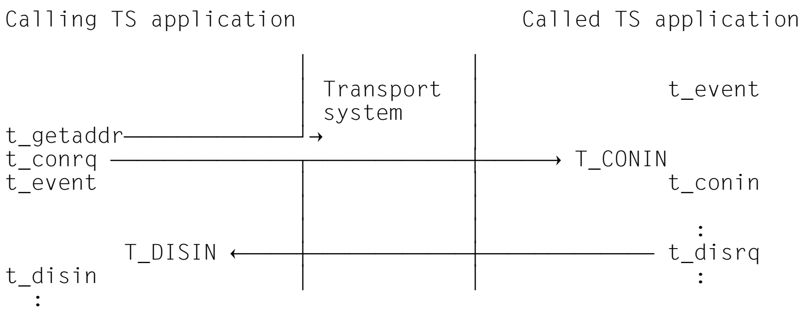
With *t_conrq()* the calling TS application may pass user data, i.e. a small quantity of data that the called TS application receives with *t_conin()*. If the called TS application then answers the connection request with *t_conrs()*, it in turn may also pass information. This is received by the calling TS application with *t_concf()*.



Exchange of user data during connection setup

Rejecting a connection request

The called TS application may also reject the connection request. The sequence is the same. The event T_CONIN must first be accepted with *t_conin()*, but instead of the call *t_conrs()* the call instead of the call *t_conrs()* the call *t_disrq()* is issued (see section entitled "Closing down a connection").



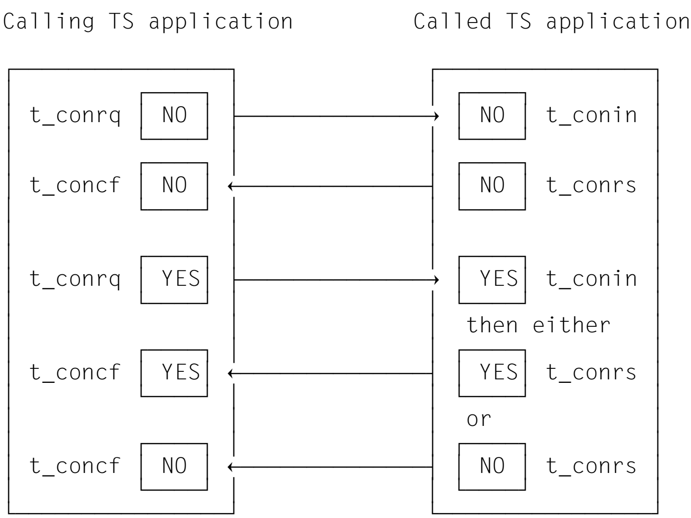
Rejecting a connection request

Agreeing on expedited data

If the transport connection to be set up permits expedited data, the TS applications may agree on its use during connection setup. This takes place as follows:

With the connection request with *t_conrq()*, the calling TS application makes a proposal, which the called TS application can only "negotiate down". This means: If the calling TS application proposes not using any expedited data, then this is settled for the connection. If, on the other hand, it proposes that expedited data be exchanged, the called TS application may accept or reject this in its connection response with *t_conrs()*. In both cases the answer is binding.

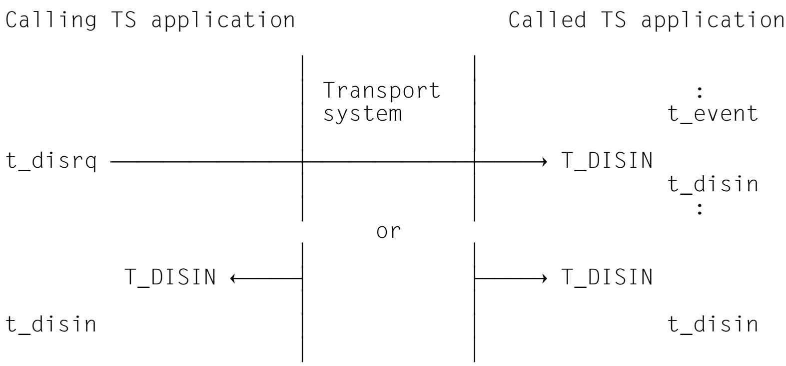
If one of the two TS applications does not agree with the result of the expedited data negotiation, it may close down the connection.



Negotiation regarding expedited data during connection setup

6.2 Closing down a connection

Either of the two communicating TS applications may call *t_disrq()* in order to close down the connection. The partner TS application then receives the event T_DISIN.
By calling *t_disin()* it accepts the disconnection. With this call it obtains the reason for the disconnection.



Closing down a connection

If the transport system provides the appropriate option, the TS application that closes down the connection may include user data with *t_disrq()*. The partner TS application receives this with *t_disin()*.

The transport system can also close down the connection. In this case, both TS applications receive the event T_DISIN, which they must fetch with *t_disin()*. Based on the reason given for the disconnection, each TS application can ascertain whether the connection was closed down by the other TS application or by the transport system.

6.3 Example of setting up and closing down a connection with ICMX

The two following program fragments show how a connection is set up.
Example 1 shows the program structure for the calling TS application.
Example 2 shows the program structure for the called TS application.

Example 1:

The TS application actively sets up a connection to the TS application "Test-application-PAS" and then closes it down.

```
#include        <stdio.h>
#include        <cmx.h>
   .
   .
#define ERROR   1
   .
   .
int     tref;                   /* Transport reference */
int     reason;                 /* Reason for disconnection */

 /* Structures for addressing */

#define PNAME   "Test_application_PAS"
char *pname = { PNAME } ;
struct t_partaddr t_partaddr;
   .
   .
/* Set up connection to the passive partner */

if ((p_partaddr = t_getaddr(pname, NULL)) != NULL)
         t_partaddr = *p_partaddr;
else {
      fprintf(stderr, ">>> ERROR 0x%x in t_getaddr\n", t_error());
      exit(ERROR);
}
if (t_conrq(&tref, (union x_address *)&t_partaddr,
              (union x_address *)&t_myname, NULL) == T_ERROR) {
      fprintf(stderr, ">>> ERROR 0x%x in t_conrq, tref 0x%x\n",
              t_error(), tref);
      exit(ERROR);
}
```

```
/* Event-driven processing:
 * t_event() waits synchronously (T_WAIT) */
 */
for (;;) {
        switch (event = t_event(&tref, T_WAIT, NULL)) {
        case T_CONCF:
           /*
            * Connection setup successful?
            */
           if (t_concf(&tref, NULL) == T_ERROR) {
                  fprintf(stderr, ">>> ERROR 0x%x in t_concf tref 0x%x\n",
                          t_error(), tref);
                  exit(ERROR);
           }
           fprintf(stderr, "Connection established to '%s'.\n",
                   pname);
           .
           .
        case T_DISIN:

            /* Disconnection by partner or system */

            if (t_disin(&tref, &reason, NULL) == T_ERROR) {
                fprintf(stderr, ">>> ERROR 0x%x in t_disin tref 0x%x\n",
                        t_error(), tref);
                exit(ERROR);
            }
            fprintf(stderr, "Received disconnect indication, tref 0x%x,
                   reason %d\n", tref, reason);
            .
            .
        }
}
 /* Disconnection */

 if (t_disrq(&tref, NULL) == T_ERROR)
            fprintf(stderr, ">>> ERROR 0x%x in t_disrq tref 0x%x\n",
                    t_error(), tref);
            exit(ERROR);
 }
 fprintf(stderr, "Connection tref 0x%x actively closed down.\n", tref);
    .
    .
```

Example 2:

The TS application waits passively for an incoming connection request, accepts the connection, and then closes it down.

```
#include        <stdio.h>
#include        <cmx.h>
   .
   .
#define ERROR   1
   .
   .
int     tref;                   /* Transport reference */
int     reason;                 /* Reason for disconnection */
/*
 * Structures for addressing
 */
struct t_myname t_myname, *p_myname;
struct t_partaddr t_partaddr;
   .
   .
   .
/* Event-driven processing:
 * t_event() waits synchronously (T_WAIT)
 */
for (;;) {
        switch (event = t_event(&tref, T_WAIT, NULL)) {

        case T_CONIN:

            /* Accept connection request */

            if (t_conin(&tref, (union x_address *)&t_myname,
                        (union x_address *)&t_partaddr, NULL) == T_ERROR) {
                fprintf(stderr, ">>> ERROR 0x%x in t_conin tref 0x%x\n",
                        t_error(), tref);
                exit(ERROR);
            }

            if (t_conrs(&tref, NULL) == T_ERROR) {
                fprintf(stderr, ">>> ERROR 0x%x in t_conrs tref 0x%x\n",
                        t_error(), tref);
                exit(ERROR);
                }
                .
                .
```

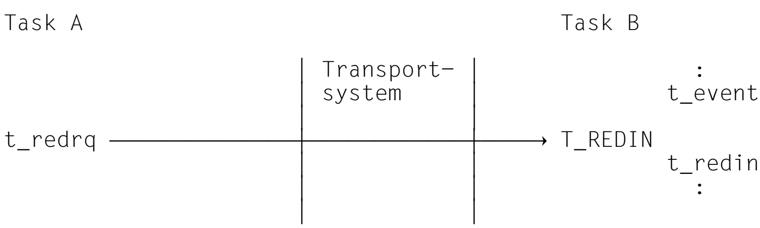
```
        case T_DISIN:
            /*
             * Disconnection by partner or system
             */
            if (t_disin(&tref, &reason, NULL) == T_ERROR) {
                fprintf(stderr, ">>> ERROR 0x%x in t_disin tref 0x%x\n",
                        t_error(), tref);
                exit(ERROR);
             }
             fprintf(stderr, "Received disconnect indication, tref 0x%x,
                   reason %d\n", tref, reason);
                .
                .
        }
}
/*
 * Disconnection
 */
if (t_disrq(&tref, NULL) == T_ERROR){
    fprintf(stderr, ">>> ERROR 0x%x in t_disrq tref 0x%x\n",
            t_error(), tref);
    exit (ERROR);
}
fprintf(stderr, "Connection tref 0x%x actively closed down.\n", tref);
    .
    .
```

6.4 Redirecting connections

Incoming connections for a local TS application are initially received by the task that first attached itself for that TS application. But in order to be able to associate particular connections with particular tasks, for example, a connection may be redirected to another task. Of course, connections set up actively may also be redirected. Both tasks must belong to the same TS application, i.e. they must have attached themselves with the same LOCAL NAME.

Sequence in redirecting a connection

Task A specifies the task ID of task B when calling *t_redrq()*. Task B receives the event T_REDIN and must initially accept the connection, with the call *t_redin()*. With this call task B is informed of the task ID of task A. If task B does not wish to have the connection, it may close it down or redirect it again, e.g. back to task A.



Redirecting a connection

With *t_redrq()* it is also possible to include user data, which task B receives when it calls *t_redin()*.

6.5 Example of redirecting a connection

The following program fragments show how a connection can be redirected and how a redirected connection is accepted.

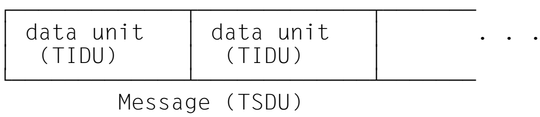
```
#include        <stdio.h>
#include        <cmx.h>
   .
   .
#define ERROR   1
   .
   .
int     tref;     /* Transport reference */
int     cpid;     /* ID of task to receive connection */
int     rpid;     /* ID of task wanting to relinquish connection */
   .
   .
/* Actively redirect connection */

if (t_redrq(&tref, &cpid, NULL) == T_ERROR) {
    fprintf(stderr, ">>> ERROR 0x%x in t_redrq tref 0x%x\n",
            t_error(), tref);
    exit(ERROR);
}
/* Accept connection redirection */

for (;;) {
        switch (event = t_event(&tref, T_CHECK, NULL)) {

        case T_REDIN:
             if (t_redin(&tref, &rpid, NULL) == T_ERROR) {
                 fprintf(stderr, ">>> ERROR 0x%x in t_redin tref 0x%x\n",
                         t_error(), tref);
                 exit(ERROR);
             }
         }
}
```

7 Transmitting data

Once a connection has been set up, the two TS applications can exchange data. Either TS application may initiate the data exchange, regardless of whether it is the calling or the called TS application.

The amount of data forming a logical unit from the point of view of the TS applications is referred to as a message, or TSDU (Transport Service Data Unit). A TSDU may be any length.

However, CMX(BS2000) can accept only a limited amount of data at any one time. This is referred to as a data unit or TIDU (Transport Interface Data Unit). The maximum length of a TIDU depends on the transport connection. This length must be queried for every connection using the *t_info()* call.



TIDU and TSDU

The logical linkage of TIDUs to form a TSDU is controlled by means of a parameter, which specifies for each TIDU in a message whether it is followed by a further TIDU or is the last one in the TSDU.

If the transport connection provides this option, and both TS applications agree to it when the connection is set up, they may also exchange expedited data. Expedited data is a small quantity of data that is given priority over normal data, i.e. expedited data never arrives later than normal data sent subsequently to the expedited data. Expedited data must always be transmitted all at once. A unit of expedited data is called an ETSDU (Expedited Transport Service Data Unit).

7.1 Sending and receiving normal data

Normal data is sent with one of the calls *t_datarq()* or *t_vdatarq()*.

Each such call sends one TIDU. *t_datarq()* is called when the TIDU to be sent is contained in one contiguous storage area. *t_vdatarq()* is called when the TIDU to be sent is located in several different storage areas.

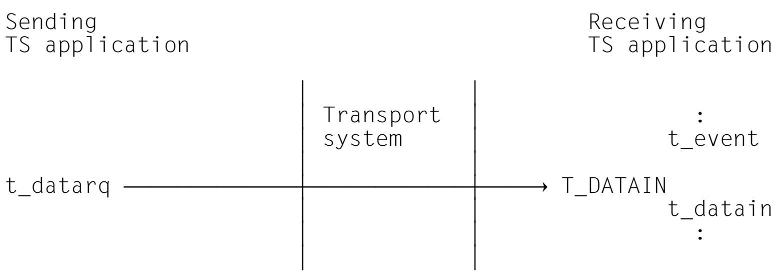
In the simplest case, data transfer proceeds as follows:

– The sending TS application passes one TIDU to CMX(BS2000) with each call.

– The receiving TS application receives the event T_DATAIN. This indicates that data has arrived.

– The receiving TS application must accept the data with the call *t_datain()* or the call *t_vdatain()*.

t_datain() and *t_vdatain()* differ, in that with *t_datain()* the data is placed into one contiguous storage area, while with *t_vdatain()* the data is placed into several different storage areas.



Transmitting normal data

If the TSDU is longer than the maximum TIDU ...

it must be broken down into TIDUs. This is done as follows:

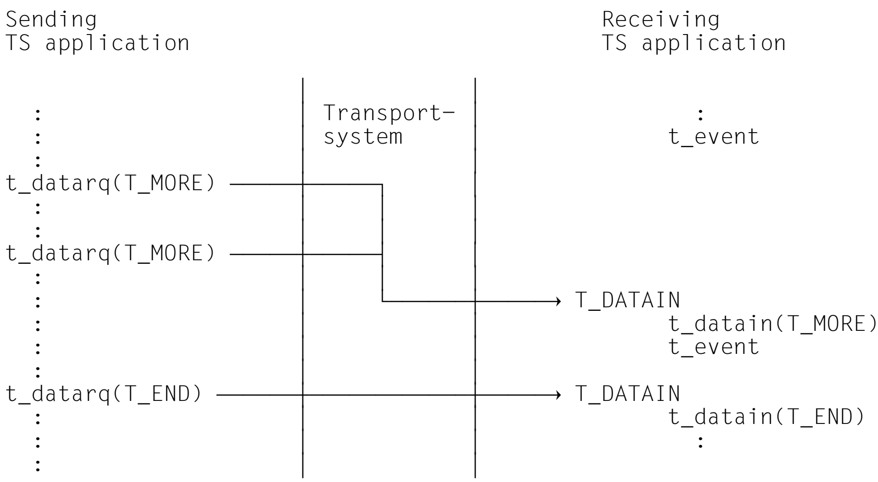
– The sending TS application determines, as sender, when the TSDU is ended. Each time a TIDU is sent with *t_datarq()* or *t_vdatarq()*, this TS application indicates in the *chain* parameter whether a further TIDU of the current TSDU is to follow (*chain* = T_MORE), or the TIDU being sent is the last one (*chain* = T_END).

– In the same way, the receiving TS application is informed with each *t_datain()* or *t_vdatain()* call by *chain* as to whether there is another TIDU to come in the current TSDU.

Each TIDU is announced by CMX(BS2000) with a T_DATAIN event. However:

There are TIDUs and TIDUs!

The length of a TIDU may be different for each of the two TS applications. Therefore it may happen that the receiving TS application will need to call *t_datain()* or *t_vdatain()* less often than the sending TS application calls *t_datarq()* or *t_vdatarq()* (or vice-versa). This is because the receiving TS application reads TIDUs in "its" length. This is represented below:



TSDU in multiple TIDUs

The value returned by t_datain() and t_vdatain()

With *t_datain()* and *t_vdatain()* you must specify a length for the incoming data to be read. If the length specified is less than the size of the TIDU for the receiver, the value returned by *t_datain()* or *t_vdatain()* will indicate the excess length of the data in the waiting TIDU.

If a TIDU has not yet been completely read, *t_datain()* or *t_vdatain()* must be called repeatedly until the TIDU has been completely read. During this time, *t_event()* may not be called, the connection may not be redirected, nor may the data flow be controlled.
Note that CMX(BS2000) does not guarantee that at the receiving TS application all TIDUs of a message will be completely filled, even when the size of a TIDU is the same for both the sending and the receiving TS application and the sending TS application sends only completely filled TIDUs.

7.2 Examples of transmitting normal data

The following program fragments show what happens when transmitting normal data via ICMX.

The TS application receives and sends data. The length of the data is limited here to one TIDU.

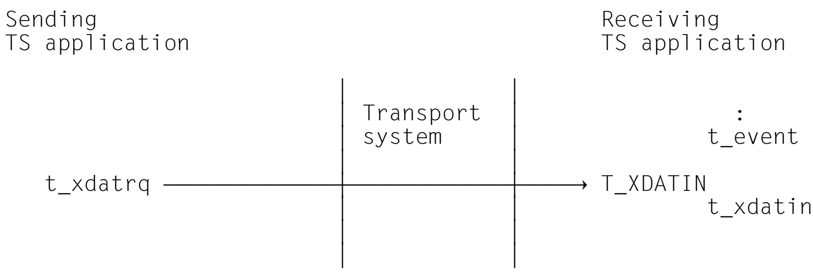
```
#include        <stdio.h>
#include        <cmx.h>
.
.
#define ERROR   1
.
.
/* Send and receive buffers */

char    e_bufpt[8000];          /* Receive buffer */
int     e_bufl;                 /* Transfer length */
char    s_bufpt[8000];          /* Send buffer */
int     s_bufl;                 /* Transfer length */

int     chain;                  /* TSDU indicator for t_datarq(), t_datain() */
int     tref;                   /* Transport reference */
 .
 .
/* Event-driven processing: */
 * t_event() waits synchronously (T_WAIT) */

for (;;) {
        switch (event = t_event(&tref, T_WAIT, NULL)) {
        .
        .
        /* Receive data; e_bufl is the TIDU length (t_info()) */

        case T_DATAIN:
            if ((rc = t_datain(&tref,e_bufpt,&e_bufl,&chain)) == T_ERROR) {
                fprintf(stderr, ">>> ERROR 0x%x in t_datain tref 0x%x\n",
                        t_error(), tref);
                exit (ERROR);
             }
             .
             .
        }
}
/* Send data; s_bufl is maximum TIDU length */

if ((rc = t_datarq(&tref, s_bufpt, &s_bufl, &chain)) == T_ERROR) {
        fprintf(stderr, ">>> ERROR 0x%x in t_datarq tref 0x%x\n",
                 t_error(), tref);
        exit(ERROR);
}
```

7.3 Sending and receiving expedited data

If the exchange of expedited data was agreed at connection setup (see the section entitled "Establishing a connection"), the TS applications may do so as follows:

Expedited data is sent with the *t_xdatrq()* call. In the simplest case the sequence is as follows:

- The sending TS application sends expedited data with a call.
- The receiving TS application receives the T_XDATIN event. This indicates that expedited data has arrived.
- The receiving TS application must accept the data with the call *t_xdatin()*.



Transmitting expedited data

The value returned by t_xdatin()

With *t_xdatin()*, a length must be specified for the incoming expedited data to be read. If the length specified is less than the amount of expedited data that has arrived, the value returned by *t_xdatin()* will then give the excess length of the waiting expedited data.

If the expedited data has not yet been completely read, *t_xdatin()* must be called repeatedly until the data has been completely read. During this time, *t_event()* may not be called, the connection may not be redirected, nor may the data flow be controlled.

```
Sending                                         Receiving
TS application                                  TS application

                        | Transport    |
  :                     | system       |                 :
  :                     |              |               t_event
t_xdatrq ---------------+--------------+-------> T_XDATIN
  :                     |              |               t_xdatin
  :                     |              |               t_xdatin
  :                     |              |               t_xdatin
  :                     |              |               t_event
  :                     |              |
  :                     |              |
```

Reading expedited data in piecemeal fashion

7.4 Flow control of normal and expedited data

If a TS application is not ready to receive data over a connection, it informs CMX(BS2000) of this with the call *t_datastop()*. CMX(BS2000) immediately stops delivering the T_DATAIN event for that connection. With one of the following *t_datarq()* calls, the communication partner will receive the return value T_DATASTOP from CMX(BS2000), and may not send any more data.

As soon as the TS application is again ready to receive data over the connection, it calls *t_datago()*. The TS application can receive data from the communication partner again. It again receives the T_DATAIN event.

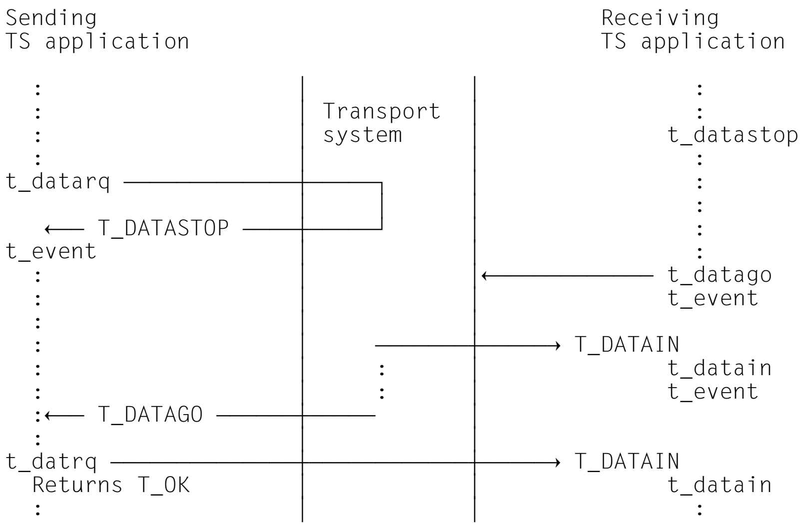
Flow control for expedited data takes place in the same way. Here the calls *t_xdatstop()* and *t_xdatgo()* are used. The corresponding events are T_XDATIN and T_XDATGO.
Note however:

When the flow of expedited data is stopped (with *t_xdatstop()*), CMX(BS2000) also implicitly stops the flow of normal data. When the flow of expedited data is then released again (with *t_xdatgo()*), the flow of normal data remains blocked! It must be expressly released (with *t_datago()*).

When the flow of normal data is released, CMX(BS2000) implicitly also releases the flow of expedited data again. Thus, after calling *t_xdatstop()*, calling *t_datago()* releases both the flow of normal data and the flow of expedited data.

What are the advantages of preventing T_DATAIN or T_XDATIN from being sent?

During this time, the TS application can issue other CMX(BS2000) calls, e.g. to set up a further connection. This would not be possible if a T_DATAIN event were waiting. If the TS application did not fetch the data, every *t_event()* call would again return the T_DATAIN event, and the TS application would not be able to receive the T_CONCF event required to set up a connection.



Data flow control at the sending end

The sending TS application receives T_DATASTOP in response to the call *t_datarq()* or *t_vdatarq()*, because the receiving TS application has stopped the data flow, or because there is a temporary resource bottleneck in CMX(BS2000) or BCAM. The data was sent, but was no longer indicated to the receiving TS application. The sending TS application must now wait with *t_event()* for the T_DATAGO event, in order to be able to send data again.

8 The ICMX program interface

This chapter describes the ICMX program interface to the CMX(BS2000) communication method CMX(BS2000). It contains:

- A summary of the functions of the ICMX interface, with details on the communication phases,
- Notes on the correct use of the functions (finite-state automata),
- Notes on the availability of the system options for the transport systems,
- Precise descriptions of the ICMX function calls, with all parameters, in alphabetical order.

8.1 Overview of the program interface

Transport Service ISO 8072

With ICMX, the present version of CMX(BS2000) provides a program interface to the connection-oriented transport service (TS) as defined in ISO 8072 within the framework of the OSI Reference Model for open systems. Therefore, in ICMX, the services T-CONNECT (connection setup), T-DISCONNECT (disconnection), T-DATA (data exchange), and T-EXPEDITED-DATA (exchange of expedited data) are defined with the primitives:

T-CONNECT.request	T-DISCONNECT.request
T-CONNECT.indication	T-DISCONNECT.indication
T-CONNECT.response	
T-CONNECT.confirmation	
T-DATA.request	T-EXPEDITED-DATA.request
T-DATA.indication	T-EXPEDITED-DATA.indication

In addition, ICMX provides local services that simplify the implementation of TS applications. These are:

T-ATTACH	Attach a TS application to CMX(BS2000)
T-DETACH	Detach a TS application from CMX(BS2000)
T-ERROR	Query errors
T-REDIRECT	Redirect a connection to another task
T-FLOWCONTROL	Flow control for normal data
T-EXPEDITED-FLOWCONTROL	Flow control for expedited data
T-EVENT	TS event check
T-INFO	Information

The TS permits two TS applications to exchange messages over a transport connection (TC). This connection-oriented communication provides for the exchange of messages without loss or duplication while maintaining the message sequence. Furthermore, by means of connection identification, the connection-oriented TS makes it possible to dispense with transferring and processing addresses in the data phase. An established TC is uniquely identified (in both end systems) by a transport reference (tref) between CMX(BS2000) and the TS application. Certain parameters that influence message transport on a TC can be negotiated between the TS applications at connection setup. To ensure that communication functions correctly, certain rules must be observed, which are described below.

ICMX is implemented as a set of C functions, which make communication between TS applications independent of the specific characteristics of the transport systems used (layers 1 - 4 in the OSI Reference Model) with regard to profiles, protocol classes, etc.

Names and addresses

Every TS application has a GLOBAL NAME. This name uniquely identifies the TS application in the network.

A TS application works exclusively with GLOBAL NAMES. a TS application obtains information from its GLOBAL NAME using CMX(BS2000) calls, e.g. the LOCAL NAME it must specify when attaching to CMX(BS2000). A TS application can use the GLOBAL NAME of a remote TS application to asecertain the TRANSPORT ADDRESS it must pass to CMX(BS2000) at connection setup.

The LOCAL NAME links the local TS application to a Transport Service Access Point (TSAP). The TRANSPORT ADDRESS of the remote TS application is required to address the Transport Service Access Point (i.e. the TS application linked to it) in the partner system.

The LOCAL NAME and TRANSPORT ADDRESS are read from the Transport Name Service.

ICMX functions for querying information from the Transport Name Service are:

t_getaddr()
Given the GLOBAL NAME of a TS application, returns its TRANSPORT ADDRESS. The TRANSPORT ADDRESS must be passed through as a parameter to the relevant ICMX call.

t_getname()
Given a TRANSPORT ADDRESS, returns the GLOBAL NAME of the TS application.

t_getloc()
Given the GLOBAL NAME of a TS application, returns its LOCAL NAME in the current end system. The LOCAL NAME must be forwarded as a parameter to the relevant ICMX call.

<*cmx.h*> defines the structures *t_myname* and *t_partaddr*.

t_myname is used by a TS application to receive (pass) the LOCAL NAME to/from CMX(BS2000) with *getloc(), t_attach()*, and *t_conrq()*; *t_partaddr* is used by a TS application to receive (pass) its TRANSPORT ADDRESS with *t_getaddr()*, *t_getname()*, *t_conin*, and *t_conrq*.

Error handling and diagnosis

All function calls return a return code. This is either T_OK, to indicate successful completion, or T_ERROR to generally indicate that an error occurred. The error check function *t_error()*, called immediately following an error, returns more detailed diagnostic information. All errors detected by CMX(BS2000) as violations of the communications rules by the TS application have specific error codes and are defined in <*cmx.h*>. The transport systems used generate no error messages; any errors result in disconnection with a corresponding reason. The reason for disconnection is obtained by the TS application when *t_disin()* is called.

The following functions return the text version of an error code returned by *t_error()*:

t_strerror()
Returns a pointer to the text string for an error code received from ICMX.

t_perror()
Calls the *ts_strerror* to ascertain the text string for an error code received from ICMX, and writes the string to stderr.

The following functions return the text for a disconnection reason returned by *t_disin()*:

t_strreason()
Returns a pointer to the text string for a disconnection reason that has been received. The reason for disconnection is passed to the TS application when *t_disin()* is called.

t_preason()
Calls *t_strreason()* to ascertain the text string for a disconnection reason that has been received with *disin()*, and writes the string to stderr.

TS applications, transport connections and tasks

A TS application is a system of programs that uses the TS, i.e. the services of CMX(BS2000). The mapping of a TS application to the task concept of the system is left up to the implementor. A TS application may organize itself into one or more (not necessarily related) tasks. In theory, the tasks may independently maintain TCs to remote TS applications. The tasks of a TS application may exchange their TCs among one another. However, at any point in time the transport reference of a TC is assigned to exactly one task. In CMX(BS2000), there is a separate local service, REDIRECT, for redirecting a TC to another task.

One task may also simultaneously control multiple TS applications. In this case, the implementation must provide for suitable coordination of the execution of the various TS applications. CMX(BS2000) supports this through its asynchronous processing mode.

Synchronous and asynchronous functions, TS events

Communications operations are by nature asynchronous: a wide variety of TS events can occur independently of the activity of a TS application. For example, a TS application may be sending data over one TC when, a disconnection indication for another TC arrives asynchronously, of which the TS application must be informed immediately.

In principle, the functions of CMX(BS2000) are asynchronous: this means, after issuing a call a TS application need not wait for a possible answer from the network. Any answer will be accepted by CMX(BS2000) when it arrives and on request, sent to the TS application as a TS event at the next opportunity.

For this, CMX(BS2000) provides the TS application with a query mechanism in two forms: synchronous (waiting) and asynchronous (checking). This query mechanism must be used appropriately by the TS application if it wishes to react quickly and properly to TS events.

With synchronous execution, the calling task is suspended until a TS event arrives. This wakes up the task, so that it can immediately process the TS event. Waiting can be limited by specifying a waiting period, or it can be cut short early by calling the function *t_wake* (wake program from *t_event*). The synchronous mechanism is useful for TS applications that maintain several TCs at a time, so that they need not poll them.

With asynchronous execution the task can check at its convenience (at the end of the processing step, for instance) whether a TS event has arrived, and handle it before continuing with the next processing step. This is useful for tasks that expect longer delays between TS events, during which times they can or must attend to other operations.

The corresponding function in CMX(BS2000) is

t_event()
: If the parameter value T_WAIT is passed, *t_event()* suspends the task until a TS event arrives, the time limit expires, or the *t_wake* function is called. If a TS event is already waiting, or there is an error, the function returns immediately with the code for the event, or T_ERROR. In contrast to CMX(SINIX), *t_event()* is not terminated automatically when a signal routine ends. Processing resumes when the function *t_wake* is called from a signal routine (contingency) or another task. *t_event()* returns with T_NOEVENT. When the time limit expires, the task resumes with the TS event T_NOEVENT. With the parameter value T_CHECK, *t_event()* always returns immediately with either the code of the TS event encountered, T_NOEVENT, or T_ERROR.

The following asynchronous TS events are defined in CMX(BS2000):

T_NOEVENT
: In the asynchronous case: No TS event present
In the synchronous case: Abort by signal or waiting time elapsed

T_CONIN
: Arrival of a connection indication from a calling TS application

T_CONCF
: Arrival of a connection confirmation from a called TS application

T_DISIN
: Arrival of a disconnect indication from a remote TS application or from CMX(BS2000)

T_REDIN
: Arrival of a redirection indication from another task of the same TS application (this TS event is local; it is an extension to the TS to make implementation of TS applications more flexible)

T_DATAIN
: Arrival of normal data from a remote TS application

T_XDATIN
: Arrival of expedited data from a remote TS application

T_DATAGO
: Removal of a block on the sending of normal data and expedited data set through flow control

T_XDATGO
: Removal of a block on the sending of expedited data set through flow control

T_SYS_EVENT

t_event() was unable to identify a signal from the CMX(BS2000) bourse mechanism as a CMX(BS2000) event.

T_ERROR

Fatal error; more detailed information is provided by the query function *t_error()*.

With each TS event, except for T_NOEVENT and T_ERROR, the TS application is also given the transport reference, so that it can react for that TC specifically to the TS event.

Some TS events must be accepted by the TS application by calling corresponding functions. Exceptions are T_ERROR, T_DATAGO, and T_XDATGO. Such function calls return additional information on the TS events. The following table lists the TS events and the corresponding functions.

TS event	Function for fetching
T_CONCF	t_concf()
T_CONIN	t_conin()
T_DATAIN	t_datain() or t_vdatain()
T_DISIN	t_disin()
T_REDIN	t_redin()
T_XDATIN	t_xdatin()

As a rule, TS events are delivered in the order in which they occur. Of course, the TS event T_XDATIN may overtake the TS event T_DATAIN, and T_DISIN may overtake T_DATAIN and T_XDATIN. In the latter case the overtaken TS events on that TC are dropped.

Attaching/detaching

Communication by a task via CMX(BS2000) is activated when the task attaches itself to CMX(BS2000). A TS application is generated when the first task attaches itself for that TS application. When this is done, a Transport Service Access Point (TSAP) is created, at which the TS is accessible. When the first task is attached, the TS application is linked to this TSAP. The TSAP is assigned the LOCAL NAME of the TS application. It thereby becomes addressable from the network. When the TS application is detached, any TCs still in existence are closed down, along with the TSAP; the task environment is dissolved, and assigned resources are released for future use.

The same task may attach itself for several TS applications at the same time (i.e. manage multiple TSAPs) and in each of these TS applications maintain multiple Transport Connection Endpoints (TCEP). Also, several tasks may attach themselves for the same TS application (use the same TSAP) and actively set up TCs or passively wait for connection indications without interfering with one another. Of course, each TCEP is assigned to exactly one task.

The following functions are used for attaching and detaching. They perform primarily local tasks. If no implicit disconnection must be performed, no information is passed to the network.

t_attach()
Attaches (the current task of) a TS application to CMX(BS2000) BCAM. When attached, the task may specify its future behavior in the TS application. The first time a task is attached CMX(BS2000) begins accepting connection indications for the TS application.

t_detach()
Detaches (the current task of) a TS application from CMX(BS2000). Any existing TCs of the task in the TS application are closed down by CMX(BS2000). If no more tasks of the TS application are attached, the TS application is no longer known to CMX(BS2000).

Connection setup, disconnection, and redirection

Before two TS applications can exchange data, a TC must be set up between them. One of the two TS applications is viewed as the calling TS application; it initiates connection setup. The other is the called TS application; it waits for requests from calling TS applications.

The calling TS application issues a connection request and receives an answer from the called TS application. The called TS application waits for a connection indication (indication of a connection request) and accepts it or rejects it. During connection setup, the TS applications negotiate certain attributes of the TC for data transmission and may exchange user data.

The TC may be closed down at any time by either of the TS applications or by CMX(BS2000). This is not negotiated between the TS applications, but instead is immediately carried out by CMX(BS2000). The other TS application (or both, if CMX(BS2000) closes down the TC) receives a disconnect indication, which may be neither answered nor averted. CMX(BS2000) indicates all errors in the transport systems by closing down the TCs involved. CMX(BS2000) does not guarantee that data still in transit at the time of the disconnection request will be delivered.

Connection redirection is a local service in CMX(BS2000) that simplifies the organization of a TS application into tasks. A task holding ca completely established TC may redirect it (depending, of course, on the state; see the diagrams on redirecting connections in the chapter entitled "Managing connections") to another task of the same TS application. The TSAP and the TCEP remain unchanged. The redirecting task loses the transport reference for the TC, whereupon the TC is no longer available. This is described in further detail in the diagram "Status of TS applications and permissible state transitions" in the section entitled "Status of TS applications and permissible state transitions".

The relevant functions are:

t_conrq()
Requests connection setup to the called TS application with the specified TRANSPORT ADDRESS. The reference to the TSAP is established via the LOCAL NAME used when the calling TS application was attached. The function returns immediately after issuing the request; the calling TS application receives a transport reference. It must then wait synchronouously or asynchronously for the answer of the called TS application (see above).

t_concf()
Accepts from CMX(BS2000) the answer of the called TS application, indicated with T_CONCF; connection setup is now complete.

t_conin()
Receives from CMX(BS2000) a connection request, indicated with T_CONIN, from the calling TS application, along with that TS application's TRANSPORT ADDRESS. The reference to the TSAP is established for the called TS application through provision of the LOCAL NAME specified when it was attached.

t_conrs()
Answers (accepts) a connection request after it has been indicated with T_CONIN and received by the TS application.

t_disrq()
Requests that a connection be closed down; this function may be called at any time by either of the TS applications; it is also used to reject a connection request (instead of accepting it) after the request has been indicated by CMX(BS2000) and received by the TS application.

t_disin()
Accepts from CMX(BS2000) the disconnect indication indicated with T_DISIN. The reason for disconnection is also passed to the TS application with this function call.

t_redrq()
Redirects a TC to a task of the same TS application; the TC is then no longer available for the redirecting task.

t_redin()
Accepts from CMX(BS2000) a connection redirection indicated with T_REDIN; the receiving task must accept it, but may immediately pass it on (return it) or close the TC down.

Data exchange and flow control

Once a connection has been set up, the initiative rests with the TS application (not with CMX(BS2000)). It may:

– send normal data and (if agreed) expedited data, or

– indicate, with *t_event()*, that it is ready to receive normal data or (if agreed) expedited data.

Data transfer is message-oriented: the TS applications exchange Transport Service Data Units (TSDU) (messages of any length) or Expedited Transport Service Data Units (ETSDU) (expedited data of limited length). Expedited data is limited to a few bytes; when transferred it is given priority over the stream of normal data and placed into separate queues. CMX(BS2000) guarantees only that expedited data will never arrive at the receiving TS application later than normal data sent subsequently. At most, one complete ETSDU may be passed to CMX(BS2000) per call.

A TSDU (which in principle may be any length) is passed to CMX(BS2000) in portions the length of one Transport Interface Data Unit (TIDU). The maximum length of a TIDU is TC-specific and must therefore be queried by CMX(BS2000) for each TC (*t_info()*). Thus, a TSDU may have to be transferred using multiple send calls. A parameter in each send call inidcates whether a further TIDU for that TSDU follows (T_MORE) or not (T_END). It cannot be determined from this how a TIDU is packed for transfer or delivery to the receiving TS application. CMX(BS2000) guarantees only that sequential joining of the TIDUs on the receiving side will reproduce the TSDU on the sending side. The maximum TIDU length may be different for the two TS applications and depends on the TC. CMX(BS2000) does not guarantee that the TIDU of a TSDU will be delivered to the TS application completely filled.

The arrival of a TIDU of a TSDU (or the arrival of an ETSDU) is indicated to the receiving TS application by means of the TS event T_DATAIN (T_XDATIN). The TS application then fetches the TIDU (ETSDU) with a corresponding function call, either completely or in piecemeal fashion. If necessary it may or must issue several similar calls in order to take in one TIDU (ETSDU) from CMX(BS2000).

The transfer of TIDUs (ETSDUs) is subject to flow control mechanisms, which can be controlled by CMX(BS2000) and the TS applications. The return code T_DATASTOP (T_XDATSTOP) returned when data is sent indicates to the sending TS application that the TIDU (ETSDU) was processed, but the flow of TIDUs (ETSDUs) has been blocked. No further TIDUs (ETSDUs) may be sent until the flow is released again. Release is indicated by means of the TS event T_DATAGO (T_XDATGO).

The receiving TS application stops and starts the flow of TIDUs (ETSDUs) by means of function calls to CMX(BS2000), which affect the sending TS application as described above.

The following functions implement data exchange and (active) flow control:

t_datarq()
Requests transfer of a TIDU from a contiguous storage area. The return code T_DATASTOP signifies that the flow is blocked; further send requests are rejected with an error until the flow is released again.

t_vdatarq()
Functions like *t_datarq*, but the TIDU can be located in multiple, non-contiguous storage areas.

t_datain()
Accepts the data of a TIDU from CMX(BS2000), placing it into a contiguous storage area, after the TIDU has been indicated with T_DATAIN. The return code specifies how much data is still contained in the current TIDU, so that a TIDU can be read in piecemeal fashion.

t_vdatain()
Functions like *t_datain*, but the TIDU can be located in multiple, non-contiguous storage areas.

t_xdatrq()
Requests transfer of an ETSDU; the return code T_XDATSTOP signifies that the flow is blocked; further send requests are then rejected with an error until the flow is released again.

t_xdatin()
Accepts the data of an ETSDU from CMX(BS2000), after it has been indicated with T_XDATIN. The return code specifies how much data is still contained in the current ETSDU, so that an ETSDU can be read in piecemeal fashion.

t_datastop()
Blocks, from the receiving side the flow of normal data over a connection; the TS event T_DATAIN will no longer be indicated for this connection by CMX(BS2000).

t_datago()
Releases, on the receiving side, the (blocked) flow of normal data and expedited data over a connection; the TS events T_DATAIN and T_XDATIN can again be indicated for the connection by CMX(BS2000).

t_xdatstop()
Blocks, on the receiving side, the flow of expedited data and normal data over a connection; CMX(BS2000) will no longer indicate the TS events T_XDATIN and T_DATAIN for this connection.

t_xdatgo()
Releases, on the receiving side, the (blocked) flow of expedited data over a connection; the event T_XDATIN can again be indicated by CMX(BS2000) for the connection.

t_wake()

awakens its own or another task from *t_event()*. The awakened task receives the return value T_NOEVENT. *t_wake()* is designed to synchronize non-CMX(BS2000) events at the CMX(BS2000) waiting point. In TU, only tasks with the same user ID can be awakened. *t-wake()* always generates a T_NOEVENT event even when the task being wakened does not call *t_event*.

Information service

The information service is a local service with which the TS application can query configuration-dependent parameter values from CMX(BS2000). The information service is implemented with the following function:

t_info()

Returns the length of a TIDU for an established TC.

8.2 States of TS applications and permissible state transitions

The sequences of operations at the ICMX program interface are represented in the following diagram by means of finite-state automata. The diagram shows the defined states that a TS application may assume during the course of communication, and the permissible transitions between these states. With the aid of the diagram, it is possible to identify permissible sequences of CMX(BS2000) calls. The diagram shows when and how the tasks of a TS application should react to certain events.

Programs that behave as shown in this state diagram are compatible with CMX(SINIX); in this situation, CMX(BS2000) and CMX(SINIX) behave in a similar manner. Results and responses differ if their behavior differs!

In the diagram, each state is represented by a rectangle with a double border. The rectangle contains the name of the state.

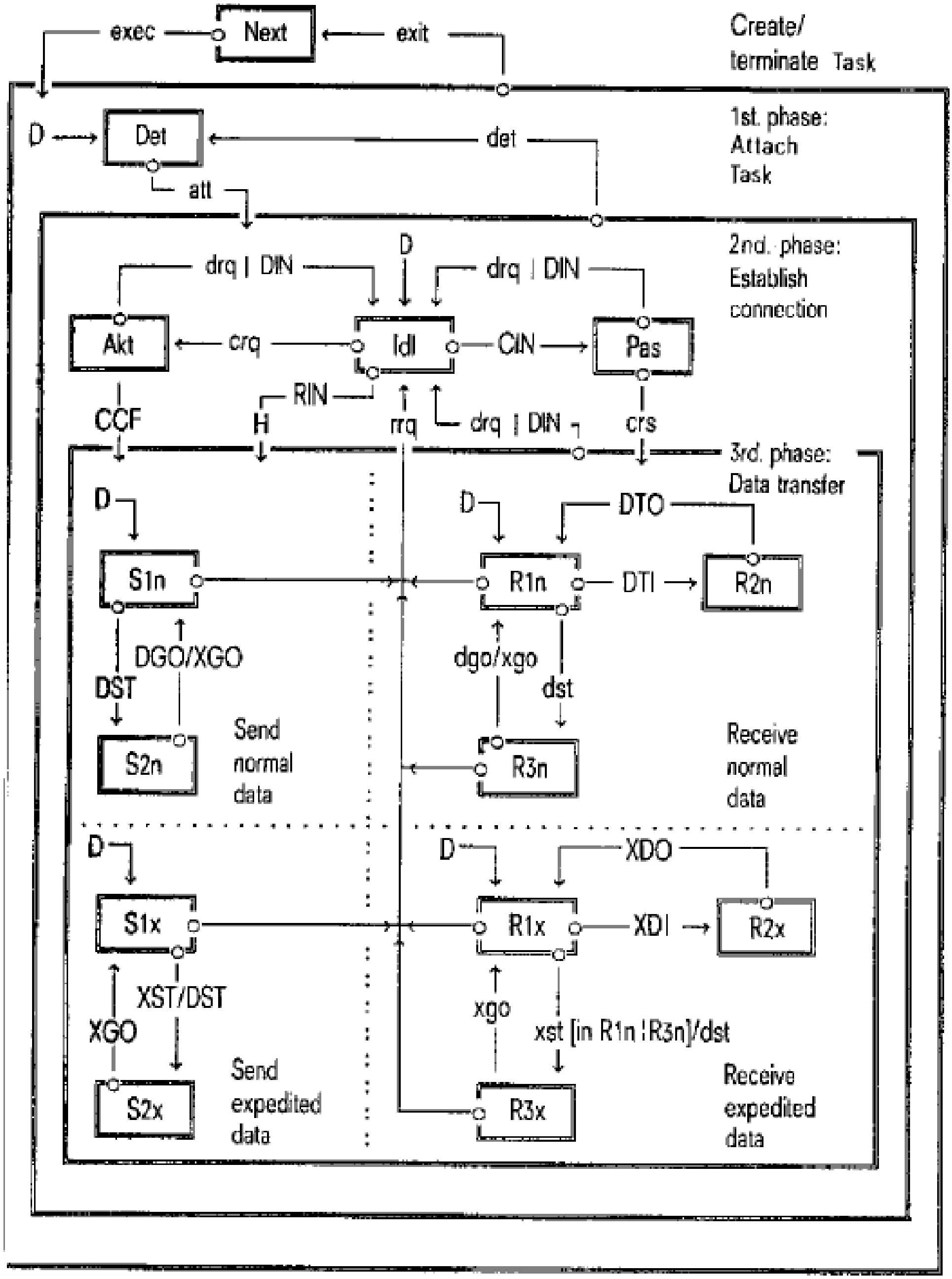
The surrounding (outer) rectangles represent the three communication phases.

1. communication phase: Attach task
   The task exists, but is not yet or no longer attached to CMX(BS2000).
2. communication phase: Connection setup
   The task is attached to CMX(BS2000), but no connection exists.
   A connection can now be set up.
3. communication phase: Data transfer
   The connection has been set up.
   The task can send and receive data.

The 3rd communication phase is subdivided by dotted lines into four subareas. These subareas are:

– Send normal data
– Receive normal data
– Send expedited data
– Receive expedited data

When it reaches this phase, the task is in exactly one state in each subarea at any given time. Only certain combinations of states in these subareas are permitted, i.e. a state transition within one subarea may cause a state transition in another subarea. The connections between the individual states in the various subareas can be seen by examining the conditions for state transitions (see below). If the exchange of expedited data has not been agreed for the connection, the task can only assume states of the top two subareas.



States of TS applications and permissible state transitions

The arrows o-C→ between the rectangles indicate the possible state transitions. C indicates the condition for making the transition from an initial state to the subsequent state (initial state o-C→ subsequent state). Transitions are possible only in the directions indicated by the arrows.

To begin with, the abbreviations used in the diagram are explained below:

Abbreviations for the states:

Nex	The task does not exist (no longer exists).
Det	The TS application is not yet attached to CMX(BS2000), or the TS application has been detached from CMX(BS2000).
Idl	Initial state for connection setup and for accepting a connection redirection, or a previously existing connection was closed down.
Act	Waiting for the event T_CONCF following a t_conrq() call (active connection setup).
Pas	A T_CONIN event has arrived (passive connection setup).
S1n	Initial state for *t_datarq()* or *t_vdatarq()*.
S2n	Normal data flow blocked.
R1n	Initial state for *t_datain()*.
R2n	T_DATAIN indicated.
R3n	T_DATAIN blocked.
S1x	Initial state for *t_xdatrq()*.
S2x	Flow of expedited data blocked.
R1x	Initial state for *t_xdatin()*.
R2x	T_XDATIN indicated.
R3x	T_XDATIN blocked.

Abbreviations for the state transition conditions

exec	Program start
exit	Program end

The state transitions below occur when a CMX(BS2000) function is called:

att	t_attach()
det	t_detach()
crq	t_conrq()
crs	t_conrs()
drq	t_disrq()
rrq	t_redrq()
dst	t_datastop()
dgo	t_datago()
xst	t_xdatstop()
xgo	t_xdatgo()

The state transitions below occur when an event is accepted:

NEV	T_NOEVENT
CIN	T_CONIN
CCF	T_CONCF
DIN	T_DISIN
RIN	T_REDIN
DTI	T_DATAIN
XDI	T_XDATIN
DGO	T_DATAGO
XGO	T_XDATGO

The following state transitions occur when certain return values are returned by CMX(BS2000) functions:

DST	T_DATASTOP returned by t_datarq() or T_vdatarq()
XST	T_XDATSTOP returned by t_xdatrq()
DTO	0 returned by t_datain() or t_vdatain() (current TIDU completely read)
XDO	0 returned by t_xdatin() (ETSDU completely read)

8.2.1 Explanations of the possible state transitions

Arrows that terminate at a surrounding rectangle indicate that normally the task first switches to the states indicated by D→.
For example, in the transition to the 3rd communication phase (data transfer) the task initially switches to the states S1n, S1x, R1n, R1x.
An exception to this is the transition RIN H→. When connection redirection occurs, this means that the receiving task assumes the states in the 3rd phase (data transfer) that the redirecting task assumed in this phase prior to the redirection.

Arrows that begin at a surrounding rectangle indicate that a transition is possible from any given state within the rectangle.

State transitions of this kind are:

– exec
  The task starts an application program that can use CMX functions.

– exit
  The application program is terminated. All connections are closed down by CMX(BS2000).

– det
  If the task calls *t_detach()* in any state, it switches to the Det state. CMX(BS2000) closes down its connections.

– drq|DIN (drq or DIN)
  If the task calls *t_disrq()* in any state during data transfer (3rd phase), or during connection setup (2nd phase), the task switches to the state Idl. The same thing happens when CMX(BS2000) indicates the T_DISIN event to the task. The existing connection is closed down or the connection request of another TS application is rejected.

State transitions within the 3rd phase (data transfer)

The following describes the connections between state transitions in the subareas of the 3rd phase. The state assumend by a task in the subarea "Send normal data" depends on its state in the subarea "Sned expedited data", and vice-versa. The state assumed by a task in the subarea "Receive normal data" depends on its state in the subarea "Receive expedited data", and vice-versa.

The following connections exist between the states of the four subareas:

DGO/XGO (DGO initiates XGO)
: The event T_DATAGO initiates T_XDATGO. Along with normal data flow, expedited data flow is released, assuming it was blocked. Thus, the state transition S2n → S1n initiates the state transition S2x → S1x.

XST/DST (XST initiates DST)
: The event T_XDATSTOP initiates the event T_DATASTOP. The state transition S1x → S2x brings about the state transition S1n → S2n. Blocking the expedited data flow causes blocking of normal data flow.

dgo/xgo (dgo initiates xgo)
: If the task calls *t_datago()* in the state R3n (T_DATAIN blocked), *t_xdatgo()* is implicitly called. The state transition R3n → R1n initiates the state transition R3x → R1x, if the task had previously assumed the state R3x.

xst[in R1n|R3n]/dst
: If the task is in the state R1x, it may call *t_xdatstop()* only if it is in the state R1n or R3n in the subarea "Receive normal data". It thereby initiates *t_datastop()*. This means the flow of expedited data can be blocked by the task only so long as no T_DATAIN is indicated. Along with the flow of expedited data, the flow of normal data is implicitly blocked (R1x → R3x initiates R1n → R3n).

8.3 System options and message length

It is important to note when creating TS applications that the system options "exchange user data when setting up and closing down a connection" and "exchange expedited data" are not supported by all transport connections. Moreover, in transport connections that support these system options, the permitted length of the user data or the expedited data unit is not always the same.

8.4 Programming notes

The primary purpose of ICMX is to make TS applications independent of the transport systems used. This allows TS applications to execute in a variety of network environments. ICMX supports this independence for TS applications that adhere to the following rules:

1) The application should make no explicit assumptions regarding the length of a TIDU or regarding the way TIDUs are packed for communication.
2) The limits defined in <*cmx.h*> for the options must never be exceeded. Please note that some transport systems do not provide certain options.
3) The TS application must only carry out name-address conversion with the *t_getloc()*, *t_getaddr()*, and *t_getname* functions. These functions are based on the name service entries in BCAM mapping.
4) CMX(BS2000) functions should not be called in contingencies. The contingencies are designed for performing asynchronous CMX processing outside the current context.
5) The program logic should be arranged in a switch/case construction, which is ideally suited for these purposes.

Example

```
Calling TS application                  Called TS application

t_attach();                             t_attach();
:                                           for (;;) {
:                                              switch (t_event()) {
t_conrq();                                     case T_CONIN:
for (;;) {                                      t_conin();
    switch(t_event()){                               :
    case T_CONCF:                                    t_conrs();
         t_concf();                                  :
         t_datarq();                           case T_DATAIN:
         :                                           t_datain();
         t_disrq();                                  :
         :                                           :
    case T_DATAIN:                                   t_datarq();
         t_datain();                                 :
           :                                         :
    case T_DISIN:                                    t_disrq();

         t_disin();                            case T_DISIN:
         :                                           t_disin();
         :                                           :
    case T_NOEVENT:                            case T_NOEVENT:
         continue;                                   continue;
    case T_ERROR:                              case T_ERROR:
         t_detach();                                 t_detach();
         exit();                                     exit();
    default:                                   default:
           :                                      :
        }                                      }
        t_detach();                             t_detach();
```

8.5 Conventions

When using ICMX the following conventions must be observed:

1) All identifiers starting with "_" (underscore) are reserved for the system software.
2) All identifiers starting with "t_" or "ts" or "Ts" are reserved for CMX(BS2000).
3) All preprocessor definitions starting with "T_" or "TS_" are reserved for CMX(BS2000).

8.6 ICMX function calls

The following pages describe the CMX(BS2000) calls in detail. Italic type in running text represents ordinary, replaceable formal parameters or the names of functions and files. Names in upper case letters (e.g. T_MSGSIZE) represent constants that have been defined in a header file (with #define).

The following conventions are used in the parameter descriptions:

-> Indicates a parameter in which CMX(BS2000) expects a value provided by the caller.

<- Indicates a parameter in which CMX(BS2000) returns a value after the call.

<> Indicates a parameter in which the caller must provide a value, which is then modified by CMX(BS2000). Modification generally only takes place if processing was successful. If it was unsuccessful, the value remains unchanged.

Of course, if a parameter involves a pointer, this marking does not refer to the pointer itself (which is always provided by the caller), but instead to the contents of the field to which the pointer points.

In sll cases, for values to be returned by CMX(BS2000) appropriate storage space must be provided by the caller, and a pointer must be passed to CMX(BS2000).

All error values that can occur with the individual calls are listed in the appendix.

t_attach
Attach a task to CMX(BS2000) (attach task)

t_attach() attaches the current task of the TS application to CMX(BS2000). If this is the first task to be attached for the TS application, the TS application (TSAP) is created. Further tasks can be attached for the same TS application. With *t_attach()*, the same task can also be attached for a number of TS applications. The LOCAL NAME of the TS application must be specified as a parameter.

However, in BS2000 the LOCAL NAME returned by *t_getloc()* does not contain the transport addresses, but rather the GLOBAL NAME with its modified syntax. Conversion to transport addresses takes place internally in BCAM when *t_attach()* is called, using the name mapping entries. The Name Service is replaced by BCAM mapping.

Privileged TS applications can be defined with BCAM mapping. Here, for each /BCMAP you must assign one privileged NEA T-sel (first character = $) and/or one privileged port number to the GLOBAL NAME of the TS applications. TS applications defined in this way can only be opened by tasks with the TSOS or NETADM privilege.

After the first successful attach operation for a TS application, CMX(BS2000) starts to accept connection requests for the application. The connection requests are always sent to the oldest task attached to the TSAP (generally the first task to open the TSAP).

Using the parameters passed by the *t_attach()* call, you can define the TS application for which the task attached itself.

```
#include <cmx.h>
int t_attach(name, opt)
    struct t_myname *name;
    union {
        struct t_opta1 opta1;
        struct t_opta2 opta2;
} *opt;
```

-> name
: Pointer to a data area with the LOCAL NAME of the TS application. The *t_getloc()* call returns the LOCAL NAME as a property of the GLOBAL NAME of the TS application.

<> opt

For the *opt* parameter, specify the value NULL or a pointer to a union with user options. If *opt* = NULL is specified, CMX(BS2000) uses the given default values. The *t_opta1* option is only supported to ensure compatibility with SINIX. It has the same effect as specifying NULL, since CMX(BS2000) does not evaluate any of the parameters.

The following structures are defined in <*cmx.h*>:

```
      struct t_opta1 {
->        int t_optnr;     /* Option no. T_OPTA1 */
->        int t_apmode;    /* Task mode */
->        int t_conlim;    /* Number of connections */
      }

      struct t_opta2 {
->        int t_optnr;     /* Option no. T_OPTA2 */
->        int t_apmode;    /* Task mode */
->        int t_conlim;    /* Number of connections */
->        int t_uattid;    /* User attachment reference */
<-        int t_attid;     /* CMX attachment reference */
<-        int t_ccbits;    /* Reserved */
<-        int t_sptypes;   /* Reserved */
      }
```

t_optnr

Option number. Specify:

T_OPTA1 in t_opta1
T_OPTA2 in t_opta2

t_apmode

t_apmode is not evaluated by CMX(BS2000).

Default value in BS2000: T-ACTIVE | T_PASSIVE | T_REDIRECT

t_conlim

t_conlim is not evaluated in BS2000. BCAM limits the maximum number of connections by TSAP and not by task. This value can be specified by the BCAM administration for the whole of BCAM.

t_uattid

In the field *t_uattid* you can pass CMX(BS2000) any user reference desired for this application. This user reference will be subsequently returned by CMX(BS2000) as an option in *t_event*, i.e. when the current task queries CMX(BS2000) regarding the arrival of an event.
This user reference makes it easier for a task that controls multiple TS applications to associate an arriving event with the appropriate attachment.

Default value if NULL specified: 0

t_attid

This field serves trace and diagnostic purposes. It is used exclusively for logging. In the *t_attid* field CMX(BS2000) returns the internal CMX(BS2000) reference to the attachment.

t_ccbits

This field is reserved and is not evaluated by CMX(BS2000).

t_sptyps

This field is reserved and is not evaluated by CMX(BS2000).

Return values

T_OK

The call was successful. The task was the first to attach itself with this name.

T_NOTFIRST

The call was successful. However, the task was not the first to attach itself for this TS application.

T_ERROR

Error. Error code can be queried using *t_error()*.

Errors

See appendix for error values.

See also

t_detach(), t_event(), t_error(), t_getloc().

t_concf
Establish connection (connect confirmation)

t_concf() accepts a T_CONCF event from CMX(BS2000) previously reported with *t_event()*. T_CONCF indicates that the called TS application has positively answered a connection request (*t_conrq()* call) of the current task.

t_concf() returns:

– The user data that the called TS application included, if the transport system used provides this option.

– The answer of the called TS application if the current task proposed the exchange of expedited data when issuing the connection request *t_conrq()*.

If the *t_concf()* call is successful the connection is established for the current task. As soon as a connection is established, the TS application (not CMX(BS2000)) has the initiative. It may:

– send normal data and (if agreed) expedited data, or

– indicate, through *t_event()*, that it is ready to receive normal data or (if agreed) expedited data, or

– redirect or close down the connection.

```
#include <cmx.h>
int t_concf(tref,opt)
    int *tref;
union {struct t_optc1 optc1;} *opt;
```

-> tref

Pointer to a field with the transport reference of the connection, passed to the current task via *t_event()*.

<> opt

For *opt*, specify the value NULL or a pointer to a union containing a structure with system options. This union is used to receive the user data that the called TS application included with its answer to the connection request.
If *opt* = NULL is specified, CMX(BS2000) discards the user data and options.
If the called TS application specified no user data and no options, CMX(BS2000) uses the given default values.

The following structure is defined in <*cmx.h*>:

```
     struct t_optc1 {
 ->      int  t_optnr;    /* Option no. */
 <-      char *t_udatap;  /* Data buffer */
 <>      int  t_udatal;   /* Length of the data buffer */
 <-      int  t_xdata;    /* Choice for expedited data */
 <-      int  t_timeout;  /* Reserved */
     };
```

t_optnr
: Option number. Specify T_OPTC1.

t_udatap
: Pointer to a data area in which CMX(BS2000) enters the user data received from the called TS application. The area must be large enough to accommodate the data received. The maximum required length depends on the transport connection being used.

  Default value if NULL specified: Undefined

t_udatal
: Prior to the call 0 or the length of the data area *t_udatap* must appear here. T_CON_SIZE is the maximum size suitable for all transport systems. T_CON_SIZE is defined in <*cmx.h*>.
  After the call, CMX(BS2000) returns in this field the number of bytes placed in *t_udatap*.

  Default value if NULL specified: 0

t_xdata
: CMX(BS2000) returns here the answer of the called TS application if the exchange of expedited data was proposed at connection setup. The answer is binding. Possible answers:

  T_YES
  : The called TS application accepts the proposal.

  T_NO
  : The called TS application rejects the proposal.

  Default value if NULL specified: T_NO

t_timeout
: This field always contains T_NO.

Return values

T_OK
: The call was successful.

T_ERROR
: Error. Error code can be queried using *t_error()*.

Errors

See appendix for error values.

See also

t_conrq(), t_error(), t_event()

t_conin
Receive connection request (connect indication)

t_conin() accepts a T_CONIN event previously reported with *t_event()*. T_CONIN indicates that a calling TS application wishes to set up a connection to the current task.

The call returns:

- the TRANSPORT ADDRESS of the calling TS application,
- the LOCAL NAME of the local TS application, and
- the user data that the calling TS application included.

Subsequently, the connection request may be answered (confirmed) with *t_conrs()* or rejected with *t_disrq()*.

```
#include <cmx.h>
int t_conin(tref, toaddr, fromaddr, opt)
int *tref;
union t_address *toaddr;
union t_address *fromaddr;
union {struct t_optc1 optc1;} *opt;
```

-> tref

Pointer to a field with the transport reference of the connection, passed to the current task via *t_event()*.

<- toaddr

Pointer to a union *t_address* in which CMX(BS2000) returns the LOCAL NAME of the called TS application that is to receive the connection.
If the current task is attached for multiple TS applications, this information can be used to associate the connection request with the correct TS application.

<- fromaddr

Pointer to a union *t_address* in which CMX(BS2000) returns the TRANSPORT ADDRESS of the calling TS application. The TRANSPORT ADDRESS can be converted to the GLOBAL NAME of the calling TS application with the aid of the call *t_getname()*.

<> opt

For *opt*, specify the value NULL or a pointer to a union containing a structure with system options.

This union is used to fetch the user data that the calling TS application specified at connection setup.
If *opt* = NULL is specified, CMX(BS2000) discards the user data.

If the calling TS application specified no user data and no options in *t_conrq()*, CMX(BS2000) returns the specified default values.

The following structure is defined in <*cmx.h*>:

```
     struct t_optc1 {
->       int  t_optnr;    /* Option no. */
<-       char *t_udatap;  /* Data buffer */
<>       int  t_udatal;   /* Length of the data buffer */
<-       int  t_xdata;    /* Choice for expedited data */
<-       int  t_timeout;  /* Inactive time */
     };
```

t_optnr
: Option number. Specify T_OPTC1.

t_udatap
: Pointer to a data area in which CMX(BS2000) enters the user data received from the calling TS application.
The area must be large enough to accommodate the user data received. The maximum length of user data required depends on the transport connection being used.
T_CON_SIZE is the maximum size suitable for all transport systems. T_CON_SIZE is defined in <*cmx.h*>.

  Default value if NULL specified: Undefined

t_udatal
: Prior to the call, 0 or the length of the data area *t_udatap* must appear here.
After the call, CMX(BS2000) returns in this field the number of bytes placed in *t_udatap*.

  Default value if NULL specified: 0

t_xdata
: In this field, CMX(BS2000) returns the proposal of the calling TS application regarding expedited data.

  Possible answers:

  T_YES
  : The calling TS application proposes exchanging expedited data.

  T_NO
  : The exchange of expedited data is ruled out by the calling TS application.

If the calling TS application proposes exchanging expedited data (T_YES), the answer of the current task in the subsequent *t_conrs()* is final.
If the calling TS application desires no expedited data (T_NO), none can be requested by the current task in the subsequent *t_conrs()*. It may then be necessary for the current task to reject the connection request with *t_disrq()*.

Default value if NULL specified: T_NO

t_timeout
This field always contains T_NO.

Return values

T_OK
The call was successful.
T_ERROR
Error. Error code can be queried using *t_error()*.

Errors

See appendix for error values.

See also

t_attach(), t_conrs(), t_conrq(), t_disrq(), t_error(), t_event(), t_getname()

t_conrq
Request connection (connection request)

t_conrq() requests the establishment of a transport connection from the local TS application to a called TS application (active connection setup).

More specifically, the effects of *t_conrq()* are:

– The called TS application receives the event T_CONIN as a connection indication, to which it must respond.
  CMX(BS2000) later indicates the answer of the called TS application to the current task in a *t_event()* call as T_CONCF or T_DISIN.

– The called TS application may be sent user data along with the connection request, if the transport system used provides this option.

```
#include <cmx.h>
int t_conrq(tref, toaddr, fromaddr, opt)
int *tref;
union t_address *toaddr;
union t_address *fromaddr;
union {
        struct t_optc1 optc1;
        struct t_optc3 optc3;
} *opt;
```

<- tref
: Pointer to a field in which CMX(BS2000) returns the connection-specific transport reference. This uniquely identifies the connection in the subsequent communication phases. It must therefore be specified with all calls that involve this connection.

-> toaddr
: Pointer to a union *t_address* with the TRANSPORT ADDRESS of the called TS application. The TRANSPORT ADDRESS is returned by the call *t_getaddr()* as a property of the GLOBAL NAME of the called TS application. It can be ascertained in advance using the *t_getaddr()* call.

-> fromaddr
: Pointer to a union *t_address* with the LOCAL NAME of the calling TS application. The same LOCAL NAME must be specified here as was specified in *t_attach()* for this TS application.

-> opt

For *opt*, specify the value NULL or a pointer to a union with system options.
This is used to specify the user data and options that the called TS application is to receive with the connection indication.
If *opt* = NULL is specified, CMX(BS2000) uses the given default values.

The following structures are defined in <*cmx.h*>:

```
    struct t_optc1 {
->      int  t_optnr;    /* Option no. */
->      char *t_udatap;  /* Data buffer */
->      int  t_udatal;   /* Length of the data buffer */
->      int  t_xdata;    /* Choice for expedited data */
->      int  t_timeout;  /* Inactive time */
    };

    struct t_optc3 {
->      int  t_optnr;    /* Option no. */
->      char *t_udatap;  /* Data buffer */
->      int  t_udatal;   /* Length of the data buffer */
->      int  t_xdata;    /* Choice for expedited data */
->      int  t_timeout;  /* Inactive time */
->      int  t_ucepid;   /* User connection reference */
    };
```

t_optnr

Option number. Specify:

T_OPTC1 in t_optc1
T_OPTC3 in t_optc3

t_udatap

Pointer to a storage area containing user data that the called TS application is to receive with the connection indication.

Default value if NULL specified: Undefined

t_udatal

Length of the user data, in bytes, to be transferred from the area *t_udatap*.
If 0 is specified for *t_udatal*, *t_udatap* is ignored.
The maximum value for *t_udatal* depends on the transport connection that is to be set up.

Default value if NULL specified: 0

t_xdata

In the *t_xdata* parameter, the current task informs the called TS application as to whether it is ready to exchange expedited data.

Permissible values are:

T_YES

Exchange of expedited data proposed.

T_NO

Exchange of expedited data ruled out.

Default value if NULL specified: T_NO

t_timeout

The *t_timeout* option is only supported to ensure compatibility with SINIX. In CMX(BS2000), there is no time monitoring of the connections. The parameter is silently ignored by CMX(BS2000).
Default value: T_NO.

t_ucepid

This field can be used to pass a freely-selectable user reference for this connection to CMX(BS2000).
This user reference can be returned to the current task by CMX(BS2000) as an option in a *t_event()* call.
If the current task is maintaining multiple connections, this mechanism enables it to associate a TS event with the appropriate connection via a user-defined attribute.
The user reference constitutes an alternative to the transport reference *tref*, defined by CMX(BS2000).

Default value if NULL specified: 0

Note

If several routes (from generation) to a TS application are available for the transport connection, the transport system itself selects a suitable one. A specific route can only be assigned by using a BCAM mapping entry for the called TS application. This route is then always used.

If the underlying protocol does not permit the exchange of connection data, this data is lost, sometimes without being reported.

The memory storage areas must be allocated with either read (**fromaddr, *toaddr, *opt, *t_udatap*) or write (**tref*) access; otherwise, the program terminates with an address error. CMX(BS2000) recognizes that user data has been specified by *t_ud<atal* > 0. *t_udatap* = NULL is permissible.

Return values

T_OK
: The call was successful.

T_ERROR
: Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*. All error values are listed in the appendix.

See also

t_attach(), t_error(), t_event(), t_getaddr().

t_conrs
Respond to connection request (connection response)

t_conrs() is used by the called TS application to accept (confirm) the connection request of a calling TS application, the connection request having been previously indicated to the current task in *t_event()* with the T_CONIN event. The current task must accept the T_CONIN event with *t_conin()* (passive connection setup) before calling *t_conrs()*. The calling TS application receives this response as connection confirmation with the T_CONCF event.

With *t_conrs()*

– user data can be sent to the calling TS application, if the transport system used provides this option;

– the connection is completely set up for the current task.

As soon as a connection has been established, the TS application (not CMX(BS2000)) has the initiative. It may:

– send both normal data and (if agreed) expedited data, or

– indicate, via *t_event()*, that it is prepared to receive normal data or (if agreed) expedited data, or

– close down or redirect the connection.

```
#include <cmx.h>
int t_conrs(tref,opt)
int *tref;
union {
        struct t_optc1 optc1;
        struct t_optc3 optc3;
} *opt;
```

-> tref
Pointer to a field with the transport reference for the connection used in the corresponding *t_conin()*.

-> opt
For *opt*, specify the value NULL or a pointer to a union with system options.
This is used by the current task to pass the user data for the calling TS application together with the response to the connection request. If *opt* = NULL is specified, CMX(BS2000) uses the given default values.

The following structures are defined in <*cmx.h*>:

```
    struct t_optc1 {
->      int  t_optnr;    /* Option no. */
->      char *t_udatap;  /* Data buffer */
->      int  t_udatal;   /* Length of the data buffer */
->      int  t_xdata;    /* Choice for expedited data */
->      int  t_timeout;  /* Inactive time */
    };

    struct t_optc3 {
->      int  t_optnr;    /* Option no. */
->      char *t_udatap;  /* Data buffer */
->      int  t_udatal;   /* Length of the data buffer */
->      int  t_xdata;    /* Choice for expedited data */
->      int  t_timeout;  /* Inactive time */
->      int  t_ucepid;   /* User connection reference */
    };
```

t_optnr

Option number. Specify:

T_OPTC1 in *t_optc1*
T_OPTC3 in *t_optc3*

t_udatap

Pointer to a storage area containing user data that the calling TS application is to receive.

Default value if NULL specified: Undefined

t_udatal

Length of the user data, in bytes, to be transferred from the area *t_udatap*.
If 0 is specified for *t_udatal*, *t_udatap* is ignored.
The maximum value for *t_udatal* depends on the transport connection.

Default value if NULL specified: 0

t_xdata

In *t_xdata* the current task responds to the proposal of the calling TS application regarding the exchange of expedited data. The proposal is passed to the task after the *t_conin()* call.

Permissible values are:

T_YES
The proposal of the calling TS application regarding expedited data is accepted.

T_NO
Expedited data is refused.
The response is binding.

If the calling TS application had ruled out the use of expedited data, the response here must be T_NO.

Default value if NULL specified: T_NO

t_timeout
With CMX(BS2000), the value of this field is always T_NO.

t_ucepid
This field can be used to pass a freely-selectable user reference for this connection to CMX(BS2000).
This user reference can be returned to the current task by CMX(BS2000) as an option in a *t_event()* call.
If the current task is maintaining multiple connections, this mechanism enables it to associate a TS event with the appropriate connection via a user-defined attribute.
The user reference constitutes an alternative to the transport reference *tref*, defined by CMX(BS2000).

Default value if NULL specified: 0

Return values

T_OK
The call was successful.
T_ERROR
Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_conin(), t_error(), t_event()

Note

The storage areas **tref, *opt, *t_udatap* must be assigned with read-access from the program; otherwise, the program will abort an address error. CMX(BS2000) recognizes that user data has been specified by *t_udatal* > 0. *t_udatap* = NULL is permissible.

t_datago
Release the flow of data (data go)

t_datago() releases the blocked flow of data on the specified connection. By means of this call, the current task informs CMX(BS2000) that it is again ready to receive data. This call also releases the flow of expedited data (if agreed) if it was (also) blocked. The call has the effect of allowing the current task to receive any pending T_DATAIN and T_XDATIN events for the specified connection.

```
#include <cmx.h>
int t_datago(tref)
int *tref;
```

-> tref

Pointer to a field with the transport reference of the connection on which the flow of data is to be released.

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_datastop(), t_xdatstop(), t_error(), t_event(), t_redin()

Note

In CMX(BS2000), data flow control at the interface is independent of data flow control on the transport connection. This means that *t_datastop()* - *t_datago()* are not detected by the other TS applications until flow control of the transport connection responds.

t_datain
Receive data (data indication)

t_datain() accepts a T_DATAIN event previously reported via *t_event()*.

By means of the *t_datain()* call, the current task receives data of a Transport Interface Data Unit (TIDU) belonging to the current Transport Service Data Unit (TSDU) of the sending TS application on the specified connection.

The maximum length of a TIDU depends on the transport connection used. It can be queried for a connection that has already been set up by means of *t_info()*.
A TIDU need not be completely full. The breakdown of a TSDU into TIDUs is purely local and does not indicate anything regarding the breakdown of the TSDU into TIDUs at the sending TS application.
Between two TIDUs of a TSDU, any other CMX(BS2000) events can occur for the same or a different connection.

When *t_datain()* is called, a contiguous data area *datap* is provided in which CMX(BS2000) enters the data of the TIDU received.

t_datain() indicates:

– (in the *chain* parameter)
  whether a further TIDU belonging to the current TSDU exists (*chain* = T_MORE) or does not exist (*chain* = T_END).
  The individual TIDUs of a TSDU are each indicated via *t_event()* with the event T_DATAIN.

– (with the return value)
  whether the current TIDU has been completely read or not.
  If the value T_OK is returned, the TIDU fits into the storage area provided. The current task has received the current TIDU in its entirety.
  If a value n > 0 is returned, only a part of the TIDU has been read. n is the number of bytes of the TIDU that have not yet been read (remaining length). In this case *t_datain()* or *t_vdatain()* must be called repeatedly until the entire TIDU has been read. Only then can other CMX(BS2000) calls be issued again, e.g. *t_event()*.

```
#include <cmx.h>
int t_datain(tref, datap, datal, chain)
int *tref;
char *datap;
int *datal;
int *chain;
```

-> tref
Pointer to a field containing the transport reference of the connection, obtained via *t_event()*.

<- datap
Pointer to a storage area in which CMX(BS2000) enters the data of the TIDU received.

<> datal
Prior to the call, a pointer must be specified for *datal* indicating a feild in which the length of *t_datap* must be entered (at least 1). Following the call, CMX(BS2000) returns in this field the number of bytes entered in the *datap* storage area. This need not be the maximum length of the TIDU.

<- chain
chain is a pointer to a field in which CMX(BS2000) returns an indicator. This indicator shows whether or not an additional TIDU belonging to the TSDU exists.

Possible values:

T_MORE
Another TIDU belonging to the TSDU follows. It will be indicated with a separate T_DATAIN event.

T_END
The present TIDU is the last of the TSDU.

Return values

T_OK
The call was successful. The TIDU was completely read.

n > 0
n bytes are still contained in the TSDU.

T_ERROR
Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_error(), t_event(), t_info(), t_vdatain()

t_datarq
Send data (data request)

t_datarq() sends the next (or only) Transport Interface Data Unit (TIDU) of a Transport Service Data Unit (TSDU) to the receiving TS application on the specified connection.

The TIDU to be sent by *t_datarq()* must be provided by the current task in a contiguous data area.

If the TSDU to be sent is longer than one TIDU, it must be transferred using several *t_datarq()* (or *t_vdatarq()*) calls in succession. Therefore, in each *t_datarq()* call, the sending task must specify in the *chain* parameter whether additional TIDUs belonging to the same TSDU follow.

The maximum length of a TIDU depends on the transport connection used. It can be queried for an established connection by means of *t_info()*.

If *t_datarq()* returns the value T_DATASTOP, the TIDU has been accepted by CMX(BS2000) but the flow of TIDUs on this connection has been blocked.

The flow of TIDUs can be blocked by:

– the receiving TS application,
  which can block the flow of TIDUs by calling *t_datastop()* or *t_xdatstop()*, or

– CMX(BS2000),
  if the local buffer is full.

If the flow of TIDUs is blocked, before further TIDUs can be sent you must wait, by means of *t_event()*, for the T_DATAGO event for the connection.

Successful termination of *t_datarq()* (T_OK) does not mean that the receiving TS application has already accepted the data.
Unsuccessful termination of *t_datarq()* (T_ERROR) always means that an error has been detected locally.

```
#include <cmx.h>
int t_datarq(tref, datap, datal, chain)
int *tref;
char *datap;
int *datal;
int *chain;
```

-> tref
  Pointer to a field with the transport reference of the connection.

-> datap
Pointer to a storage area containing the TIDU to be sent.

-> datal
Pointer to a field containing the number of bytes to be sent from the storage area *datap*. You must specify at least 1, and at most the maximum length of a TIDU.

-> chain
Pointer to an indicator used by the task to indicate whether there is an additional TIDU belonging to the TSDU.

Possible values:

T_MORE
Another TIDU belonging to the TSDU follows.

T_END
The present TIDU is the last of the TSDU.

Return values

T_OK
The call was successful; further TIDUs may be sent immediately.

T_DATASTOP
The call was successful, but further TIDUs may not be sent until the event T_DATAGO has arrived for this connection.

T_ERROR
Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_datastop(), t_error(), t_event(), t_info(), t_vdatarq(), t_xdatstop()

Note

It is forbidden to continue to send data "on spec" after T_DATASTOP. Although CMX(BS2000) will not prevent this, offenders must realize that this will lead to the arrival of either too many or already invalid T_DATAGO events.

The storage area **datap* must be assigned with read-access from the program; otherwise, the program will abort with an address error. Null is a valid address for *datap*.

t_datastop
Stop the flow of data (data stop)

t_datastop() blocks data indication on the specified connection.

In particular, the effects of *t_datastop()* are:

- The current task tells CMX(BS2000) that, until further notice, it is not ready to receive data for this connection. However, a T_DATAIN event that has already been indicated must be responded to first.
- The current task no longer receives the T_DATAIN event for the specified connection. However, while the data display is blocked, it may call other CMX(BS2000) functions, e.g. to set up, close down, or redirect an additional connection.
- The sending TS application receives the return value T_DATASTOP when it calls *t_datarq()*. It may not send any more data.

The flow of data is released with *t_datago()*.
Expedited data is not affected by *t_datastop()*.

```
#include <cmx.h>
int t_datastop(tref)
int *tref;
```

-> tref
Pointer to a field with the transport reference of the connection.

Return values

T_OK
The call was successful.
T_ERROR
Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can queried by calling *t_error*.
All error values are listed in the appendix.

See also

t_datarq(), t_datago(), t_event(), t_xdatstop()

Note

Unlike CMX(SINIX), CMX(BS2000) accepts the call *t_datastop()* even when T_DATAIN has been indicated for this connection, and data has not been fully retrieved. However, data must still be fetched using a *t_datain()* or *v_datain()* call.

t_detach
Detach a task from a TS application (detach task)

t_detach() detaches the current task for the TS application specified in the parameter *name*. If connections still exist for this task, they are implicitly closed down. Normally however, all connections for this task should be closed down with *t_disrq()* before calling *t_detach()*.

When the last task of a TS application detaches itself, the TS application ceases to exist. Connection requests for that TS application will then no longer be accepted.
If further tasks are attached for this TS application, the task that attached itself after the current task will receive all subsequent connection requests.

```
#include <cmx.h>
int t_detach(name)
struct t_myname *name;
```

-> name
: Pointer to a structure *t_myname* with the LOCAL NAME of the TS application. The same LOCAL NAME must be specified as with *t_attach()*.

Return values

T_OK
: The call was successful.

T_ERROR
: Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_attach(), t_error()

t_disin
Accept disconnection (disconnection indication)

t_disin() accepts a T_DISIN event previously reported with *t_event()*. T_DISIN indicates that the connection has been closed down.

t_disin() specifies whether the remote TS application or the transport system initiated T_DISIN event.

In addition, *t_disin()* returns:

– the user data sent by the remote TS application, if the T_DISIN event was initiated by the remote TS application and if the transport connection used provides this option;

– the reason for closing the transport connection, if the T_DISIN event was initiated by CMX(BS2000) or by the transport system.
The reason for the disconnection is returned by *t_disin()* in hexadecimal form. The readable text form of the code can be obtained with the aid of *t_preason()* or *t_strreason()*.

```
#include <cmx.h>
int t_disin(tref, reason, opt)
int *tref;
int *reason;
union {struct t_optc2 optc2;} *opt;
```

-> tref
Pointer to a field containing the transport reference of the connection.

<- reason
Pointer to a field in which CMX(BS2000) enters the reason for the disconnection.

Possible values:

T_USER
The connection was closed down by the remote TS application.

other
The connection was closed down by CMX(BS2000) or the transport system. The reasons for disconnection are described later in this section.

<> opt
For *opt*, specify the value NULL or a pointer to a union containing a structure with system options.
This union can be used to check the user data that the remote TS application specified when closing down the connection.
If *opt* = NULL is specified, CMX(BS2000) discards the user data.

If the remote TS application did not specify any user data, CMX(BS2000) returns the specified default values. The transfer of user data when disconnecting is not guaranteed and depends on the underlying transport connection.

The following structure is defined in <*cmx.h*>:

```
     struct t_optc2 {
->        int  t_optnr;    /* Option no. */
<-        char *t_udatap;  /* Data buffer */
<>        int  t_udatal;   /* Length of the data buffer */
      };
```

t_optnr

Option number. Specify T_OPTC2.

t_udatap

Pointer to a data area in which CMX(BS2000) enters the user data received from the remote TS application. The area must be large enough to accommodate the user data received. The maximum length of user data required depends on the transport connection being used.
T_DIS_SIZE is the maximum size suitable for all transport systems.

Default value if NULL specified: Undefined

t_udatal

Prior to the call 0 or the length of the data area *t_udatap* must appear here.
After the call, CMX(BS2000) returns in this field the number of bytes placed in *t_udatap*.

Default value if NULL specified: 0

Return values

T_OK
The call was successful.

T_ERROR
Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be can be queried by calling*t_error()*.
All error values are listed in the appendix.

See also

t_detach(), t_disrq(), t_event(), t_preason(), t_strreason()

Reasons for disconnection

The disconnection grounds given in *reason* have the following significance. The given symbolic values are defined in *cmx.h*; where any doubt arises the numerical value defined in *cmx.h* is valid.

```
T_USER         0 at the request of the partner the connection was closed down
T_RUNKNOWN   256 disconnection from the remote transport system, no reason given.
T_RPERMLOST  261 disconnection by the administration of the transport system.
T_RSYSERR    262 disconnection from the remote transport system due to network errors.
T_RCONGEST   385 disconnection from the remote transport system due to resource scarcity.
T_RNOCONN    392 connection of the network connection to the remote transport system
                 rejected.
T_RLCONGEST  448 disconnection from the local transport connection due to resource
                 scarcity.
T_RLPROTERR  464 disconnection from the local transport system due to transport protocol
                 error (System error).
T_RLPERMLOST 481 disconnection by the administration of the transport system.
```

Note

The storage area **datap* must be assigned with read-access (**tref*) or write-access (**reason*, **opt*, **t_udatap*) from the program; otherwise, the program will abort with an address error. CMX(BS2000) recognizes that user data has been specified by *t_udatal > 0*, *t_udatap* = NULL is permissible!.

When closing down connections, *tref* is invalid for all calls at the time of disconnection. *tref* remains valid for *t_event(tref) and t_disin(tref)* only.

t_disrq
Close down connection (disconnection request)

t_disrq() closes down the specified connection, or rejects the connection indication of a calling TS application. In both cases, the remote TS application receives a disconnect indication with the reason T_USER.

Either partner may close down the connection, regardless of which one actively set it up.

Along with the disconnection, the remote TS application may be sent user data, if the transport connection provides this option.

The *t_disrq()* call may overtake data that is still in transit. This data is then lost.

#include <**cmx.h**>
int t_disrq(tref, opt)
int *tref;
union {struct t_optc2 optc2;} *opt;

-> tref

Pointer to a field containing the transport reference of the connection to be closed down.

-> opt

For *opt*, specify the value NULL or a pointer to a union containing a structure with system options.

This union is used to specify the user data that the remote TS application is to receive along with the disconnection indication.
If *opt* = NULL is specified, CMX(BS2000) uses the default values specified.

The following structure is defined in <*cmx.h*>:

```
    struct t_optc2 {
->      int  t_optnr;    /* Option no. */
->      char *t_udatap;  /* Data buffer */
->      int  t_udatal;   /* Length of the data buffer */
     };
```

t_optnr

Option number. Specify T_OPTC2.

t_udatap

Pointer to a storage area containing user data to be received by the remote TS application.

Default value if NULL specified: Undefined

t_udatal

Length of the user data to be passed from the storage area *t_udatap*.
If *t_udatal* = 0 is specified, *t_udatap* is ignored. The maximum value for *t_udatal* depends on the transport connection.

Default value if NULL specified: 0

Return values

T_OK
The call was successful.
T_ERROR
Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_detach(), t_disin(), t_event(), t_error()

t_error
Error diagnosis (error)

t_error() returns **diagnostic information** when another CMX(BS2000) call returns T_ERROR or NULL.

The possible error messages for calls to the ICMX program interface are generated either in the CMX(BS2000) library functions in the user process or in the operating system. A distinction must be made between messages generated in CMX(BS2000) itself and those resulting from operating system calls in CMX(BS2000). The error messages generated by CMX(BS2000) are returned by *t_error()* in hexadecimal form. These error codes can be converted to *t_perror()*. *t_strerror()* returns a pointer to a static area that contains the readable text form of an error message.
t_perror() writes the readable text form of an error message to *stderr*.

The CMX(BS2000) error messages are described in the appendix.

```
#include <cmx.h>
int t_error()
```

Return values

See appendix, starting on page 159.

Files

<cmx.h> – Global CMX(BS2000) definition file

See also

t_perror(), t_strerror

t_event
Await or query event (event)

t_event() determines whether a CMX(BS2000) event has arrived for the current task.

The parameter *cmode* specifies the processing mode of *t_event()*. *t_event()* can:

- **synchronously** wait for a CMX(BS2000) event for the current task to arrive. While waiting, the task is suspended.
  In BS2000, waiting cannot be interrupted using signals; this is only possible using the *t_wake* signal routine. A time limit for synchronous waiting may be specified in the *opt* options. If no event arrives within this waiting period, waiting is terminated.
- **asynchronously** check whether a CMX(BS2000) event for the current task has arrived. The function always returns immediately to the current task.

Along with the appropriate event, *t_event()* returns:

- the transport reference of the connection involved, to permit the event to be associated with the appropriate connection (*tref* parameter),
- event-specific additional information, if this has been specified in the *opt* options.

In addition, *t_event()* permits CMX(BS2000) to signal the arrival of more data for a connection, if data indications for the connection have not been explicitly blocked via *t_datastop()* or *t_xdatstop()*. If a T_DATAIN or T_XDATIN event is indicated for a task, the connection involved may not be redirected.
More importantly, *t_event()* may not be called again until the current task has accepted the indicated data with *t_datain()*, *t_vdatain()* or *t_xdatin()*.

If several events are present for a connection, they are indicated one after another in the order in which they arrived.

Exceptions:

- A T_XDATIN event (expedited data received) may overtake T_DATAIN events (normal data received) without destroying them.
- A T_DISIN event (disconnection indication) may overtake T_DATAIN and T_XDATIN events for the connection involved and thus destroy them.
  The data that T_DATAIN/T_XDATIN was to have indicated is lost.

#include <cmx.h>
int t_event(tref, cmode, opt)
int *tref;
int cmode;
union {struct t_opte1 opte1;} *opt;

<- tref

Pointer to a field in which CMX(BS2000) returns the connection-specific transport reference. The transport reference specifies the connection to which the event belongs. For the events T_NOEVENT and T_ERROR the contents of *tref* are undefined.

-> cmode

cmode is used to specify whether *t_event()* is to synchronously wait for an event or is to asynchronously check whether an event has arrived.

Possible values:

T_WAIT (synchronous processing)

The current task is suspended until a TS event arrives, the specified waiting time elapses (*t_timeout* parameter in *opt*) or *t_wake* is called for this task. In the last two cases the event T_NOEVENT is returned. The call can be terminated by calling *t_wake* from a signal routine (contingency) or from another task with the same user ID.

T_CHECK (asynchronous processing)

The current task checks whether a TS event is waiting.

If a TS event is waiting for the current task, the event is returned to the task.
If no event is waiting, the event T_NOEVENT is returned to the task.

-> opt

For *opt*, you may specify NULL or a pointer to a union *t_optel* containing a structure with system options.
If NULL is specified, CMX(BS2000) uses the defined default values.
The following structure is defined in <*cmx.h*>:

```
    struct t_opte1 {
->      int  t_optnr;    /* Option no. */
<-      int  t_attid;    /* CMX attachment reference */
<-      int  t_uattid;   /* User attachment reference */
<-      int  t_ucepid;   /* User connection reference */
->      int  t_timeout;  /* Time limit for T_WAIT */
<-      int  t_evdat;    /* Event-specific information */
    };
```

t_optnr

Option number. Specify T_OPTE1.

t_attid

In *t_attid*, *t_event()* returns the internal CMX(BS2000) reference for the attachment involved.
The CMX(BS2000) reference is also returned by CMX(BS2000) as an option in *t_attach()*. It serves only trace and diagnostic purposes and is used exclusively for logging.

t_uattid

In *t_uattid*, *t_event()* returns the user reference for the attachment involved.
The user reference is passed to CMX(BS2000) as an option in *t_attach*. This enables a task that controls multiple TS applications to associate a TS event with the appropriate attachment of a TS application.

t_ucepid

In *t_ucepid*, *t_event()* returns the user reference for the connection involved for the TS events T_CONCF, T_DATAIN, T_XDATIN, T_DATAGO, T_XDATGO and T_DISIN.
The user reference is passed to CMX(BS2000) in *t_conrq()*, *t_conrs()* or *t_redin()*. This enables a task that maintains multiple connections to associate a TS event with the appropriate connection. This feature, selected by the user, constitutes an alternative to the transport reference *tref*, defined by CMX(BS2000).

t_timeout

With *cmode* = T_WAIT:

For *t_timeout* a waiting period may be specified during which *t_event()* is to synchronously wait for an event.
With *cmode* = T_CHECK:
Any value specified for *t_timeout* is ignored.

Possible specifications for *t_timeout*:

T_NOLIMIT

No waiting period is defined. The task waits (without time limit) until an event arrives or *t_event()* is terminated by *t_wake()*.

T_NO

The task does not wait. It resumes immediately with any TS event present or with T_NOEVENT (corresponds to *cmode* = T_CHECK).

n > 0

The task waits n seconds for the arrival of a CMX event. Accuracy is +/- 60 sec. If no CMX event for the waiting task arrives within this time period, the task resumes with the event T_NOEVENT.

Default value if NULL specified: T_NOLIMIT

t_evdat
: Here, CMX(BS2000) returns event-specific additional information.

  Possible information:

  With the events T_DATAIN and T_XDATIN the length of the indicated data is specified here.
  With the other TS events, including T_NOEVENT, the additional information is undefined.

Return values

T_CONIN
: This event indicates that a calling TS application wishes to set up a connection to the current task. This connection indication must first be fetched with *t_conin()*, then confirmed with *t_conrs()* or rejected with *t_disrq()*.

T_CONCF
: This event indicates that the called TS application has responded positively to a connection request of the current task.
  This connection setup confirmation must be fetched with *t_concf()*.

T_DATAIN
: This event indicates that data has been received via the connection specified in *tref*. The data must be fetched with *t_datain()* or *t_vdatain()*. CMX(BS2000) does not indicate this event for a connection so long as data flow on it is blocked, i.e. when the receiving task has issued *t_datastop()* for it.

T_DATAGO
: The local TS application may resume sending data on the connection specified in *tref*. Possible reaction: *t_datarq()* or *t_vdatarq()*.
  The event T_DATAGO also permits the local TS application to resume sending expedited data on this connection, assuming the sending and receiving of expedited data was agreed at connection setup.

T_DISIN
: This event indicates disconnection of the connection specified in *tref*.
  This disconnect indication must be fetched with *t_disin()*.

T_ERROR
: Error. Query error code using *t_error()*.

T_NOEVENT

This event means:

If cmode = T_CHECK

No event waiting.

If cmode = T_WAIT

Wait status of the task terminated, either by signal or because the specified waiting period elapsed. No TS event arrived.

The contents of *tref* are undefined.

T_REDIN

This event indicates that another task of the same TS application has redirected a connection to the current task.
The connection redirection must be fetched with *t_redin()*.

T_XDATIN

This event indicates that expedited data has been received on the connection specified in *tref*. The data must be fetched with *t_xdatin()*.

This event is indicated only:

– if the exchange of expedited data was agreed at connection setup, and

– while the flow of expedited data on the connection is not blocked. The flow of expedited data is blocked when the receiving task has issued *t_xdatstop()* for the connection.

T_XDATGO

With this event CMX(BS2000) indicates that the task may resume sending expedited data on the connection specified in *tref*.
Possible reaction: *t_xdatrq()*.

CMX(BS2000) indicates this event only if the exchange of expedited data was agreed at connection setup.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_attach(), t_concf(), t_conin(), t_datain(), t_datago(), t_datastop(), t_disin(), t_error(), t_redin(), t_vdatain(), t_xdatin(), t_xdatgo(), t_xdatstop(), t_wake()

Note

An event which is indicated must be fetched immediately with the appropriate call.

If an indicated event is not fetched with *t_event()*, it is reindicated by CMX(BS2000) with the next *t_event()*. If the message indicated was only partially fetched with *t_atain()*, *t_vdatain()*, or *t_xdatin*, the next *t_event* call results in T_ERROR (T_WSEQUENCE).

With *t_datarq(), t_vdatarq()*, and *t_xdatrq()*, CMX(BS2000) does not check whether an event which has just been indicated but not yet fetched is present for this *tref*. For instance with *t_event*(T_REDIN) the user must insure that *t_redin()* is called before *t_datarq()*.

If either T_DATASTOP or T_XDATSTOP arrives when sending the return code, sending may only resume once the events T_DATAGO or T_XDATGO have been indicated for this connection with *t_event()*

t_getaddr
Query TRANSPORT ADDRESS (get address)

t_getaddr() ascertains the TRANSPORT ADDRESS of a remote TS application.

For the parameter *globname*, specify the GLOBAL NAME of the TS application.
t_getaddr() returns a pointer to a static area containing the TRANSPORT ADDRESS of the TS application.

This static area is overwritten at each call. If the contents of the area must be saved, the caller must copy the area.
The length of the area to be copied is obtained from the length field *t_palng* defined in *struct t_partaddr*.

```
#include <cmx.h>
struct t_partaddr *t_getaddr(globname, opt)
char *globname;
char *opt;
```

-> globname

For this parameter, specify the GLOBAL NAME of the TS application whose TRANSPORT ADDRESS you wish to obtain.
The GLOBAL NAME is to be specified as a NULL-terminated string in the form

"NP5.NP4.NP3.NP2.NP1"

The items NPi (i=1,2,3,4,5) represent the name parts of the hierarchically-structured GLOBAL NAME. NP5 is name part[5], i.e. the name part at the lowest hierarchical level. NP1 is name part[1], i.e. the highest name part in the hierarchy. The remaining name parts must be specified in increasing hierarchical order from left to right.

If one of the name parts for a particular GLOBAL NAME has no value (e.g. NP4), and this name part is followed by another name part that is higher in the hierarchy (e.g. NP3), the separator (.) from the name part with no value must be specified.
A series of separators appearing at the end of the value of *globname* may be omitted.
The GLOBAL NAME is then specified as follows: "NP5..NP3"

At least one of the name parts NPi must be specified.

If the separator character (.) is a component of a name part, it must be represented as \. (backslash period).

Examples:

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

   Specification for *globname*:
   "G\.MEIER.DF1.MCH-P.SIEMENS-AG.D"

2. GLOBAL NAME:
   Name part[2] = BU&B
   Name part[5] = PENCILPUSHER

   Specification for *globname*:
   "PENCILPUSHER...BU&B"

-> opt
The value of *opt* must be NULL.

Return values

If the call was successful, *t_getloc()* returns a pointer to a storage area containing the TRANSPORT ADDRESS

In the case of an error *t_getloc()* returns a NULL pointer. Ther error code can be queried using *t-error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_error()

t_getloc
Query LOCAL NAME (get local name)

t_getloc() ascertains the LOCAL NAME of a TS application.

For the parameter *globname*, specify the GLOBAL NAME of the TS application.
t_getloc() returns a pointer to a static area containing the LOCAL NAME of the TS application.

This static area is overwritten at each call. If the contents of the area must be saved, the caller must copy the area.
The length of the area to be copied is obtained from the length field *t_mnlng* defined in *struct t_myname*.

```
#include <cmx.h>
struct t_myname *t_getloc(globname, opt)
char *globname;
char *opt;
```

-> globname

For this parameter, specify the GLOBAL NAME of the TS application whose LOCAL NAME you wish to obtain.
The GLOBAL NAME is to be specified as a NULL-terminated string in the form

"NP5.NP4.NP3.NP2.NP1"

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

If one of the name parts for a particular GLOBAL NAME has no value (e.g. NP4), and this name part is followed by another name part that is higher in the hierarchy (e.g. NP3), the separator (.) from the name part with no value must be specified. A series of separators appearing at the end of the value of *globname* may be omitted.
The GLOBAL NAME is then specified as follows: "NP5..NP3"

At least one of the name parts NPi must be specified.

If the separator character (.) is a component of a name part, it must be represented as \. (backslash period).

Examples:

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

   Specification for *globname*:
   "G\.MEIER.DF1.MCH-P.SIEMENS-AG.D"

2. GLOBAL NAME:
   Name part[2] = BU&B
   Name part[5] = PENCILPUSHER

   Specification for *globname*:
   "PENCILPUSHER...BU&B"

-> opt
The value of *opt* must be NULL.

Return values

If the call was successful, *t_getloc()* returns a pointer to a storage area containing the LOCAL NAME. In case of error, *t_getloc()* returns a NULL pointer. The error code can be queried using *t_error()*

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_error()

t_getname
Query GLOBAL NAME (get name)

Given the TRANSPORT ADDRESS of a remote TS application, *t_getname()* ascertains its GLOBAL NAME from TS directory 1.

The TRANSPORT ADDRESS of the TS application must be specified by the caller in the parameter *addr*.

t_getname() returns a pointer to a static area containing the GLOBAL NAME of the TS application.

This static area is overwritten at each call. If the contents of the area must be saved, the caller must copy the area.

The GLOBAL NAME is returned by CMX(BS2000) as a NULL-terminated string in the form

NP5.NP4.NP3.NP2.NP1

The items NPi (i=1,2,3,4,5) represent the name parts of the GLOBAL NAME. NP5 is name part[5], i.e. the name part at the lowest hierarchical level. NP1 is name part[1], i.e. the highest name part in the hierarchy. Teh remaining name parts are specified in increasing hierarchical order from left to right.

If one of the name parts for a particular GLOBAL NAME has no value (e.g. NP4), and this name part is followed by another name part that is higher in the hierarchy (e.g. NP3), the separator (.) from the name part with no value is nevertheless returned.
A series of separators appearing at the end of the value of *globname* is omitted.
The GLOBAL NAME is then specified by CMX(BS2000) as follows: "NP5..NP3"

If the separator character (.) is a component of a name part, it is represented as
\. (backslash period).

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

-> addr
  Pointer to a storage area with the TRANSPORT ADDRESS

-> opt
  The value of *opt* must be NULL.

Return values

If the call was successful, *t_getname()* returns a pointer to a storage area containing the GLOBAL NAME.
In case of error, *t_getname()* returns a NULL pointer. The error code can be queried using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_error()

t_info
Query information on CMX(BS2000) (information)

t_info() informs CMX(BS2000) of the length of a Transport Interface Data Unit (TIDU) for the specified connection.

```
#include <cmx.h>
int t_info(tref, opt)
int *tref;
union {struct t_opti1 opti1;} *opt;
```

-> tref
: Pointer to a field with the transport reference of the connection.

<> opt
: Pointer to a union with user options.

The following structure is defined in <*cmx.h*>:

```
   struct t_opti1 {
->      int t_optnr;     /* Option no. */
<-      int t_maxl;      /* TIDU length */
   };
```

t_optnr
: Option number. Specify T_OPTI1.

t_maxl
: In this field CMX(BS2000) enters the length of a TIDU.
This value specifies the maximum number of bytes that can be sent to CMX(BS2000) or received from CMX(BS2000) per call when transferring data over this connection.

Return values

T_OK
: The call was successful.

T_ERROR
: Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

t_perror
Output CMX(BS2000) error message in decoded form

t_perror() decodes CMX(BS2000) error messages passed to the task in hexadecimal form by CMX(BS2000) when *t_error()* is called. *t_perror()* writes the plain text form of the CMX(BS2000) error message specified in *code* to the standard error output *stderr*.

In the *s* parameter an additional explanatory text may be specified, e.g. an indication of the CMX(BS2000) call and TS application to which the error refers.

Format of output from *t_perror()*:

t_perror() first writes the text specified with *s* (if s != NULL), then : (colon) and \n (newline). This is followed by the plain text form of the CMX(BS2000) error message passed- This text consists of the error symbols, as defined in <*cmx.h*>. Each error symbol is preceded by \t.

```
#include <cmx.h>
void t_perror(s, code)
char *s;
int code;
```

-> s
: Pointer to a storage area containing text that is to precede the readable text form of the error message, or the value NULL.

-> code
: For *code*, specify the representation of the error message that was passed to the task by CMX(BS2000) when *t_error()* was called.

See also

t_error(), t_strerror()

t_preason
Decode and output reasons for disconnection

t_preason() decodes reasons for disconnection passed to the task in hexadecimal form when *t_disin()* is called.
t_preason() writes the plain text form of the reason for disconnection specified in *reason* to the standard error output *stderr.*.

In the *s* parameter an additional explanatory text may be specified, e.g. an indication of the connection or TS application to which the output refers.

Format of output from *t_preason()*:

t_preason() first writes the text specified with *s* (if s != NULL), then : (colon) and \n (newline). This is followed by the plain text form of the disconnection reason passed. This text consists of the symbol for the disconnection reason, as defined in <*cmx.h*>. The symbol for the disconnection reason is preceded by \t.

```
#include <cmx.h>
void t_preason(s, reason)
char *s;
int reason;
```

-> s
: Pointer to a storage area containing text that is to precede the plain text form of the disconnection reason, or the value NULL.

-> reason
: For *reason*, specify the representation of the disconnection reason that was passed to the task by CMX(BS2000) when *t_disin()* was called.

See also

t_disin(), t_strreason()

t_redin
Accept redirected connection (redirection indication)

t_redin() accepts a T_REDIN event previously reported with *t_event()*. T_REDIN indicates that another task of the same TS application has redirected a connection to the current task.

The event T_REDIN **must** be accepted with *t_redin()*. If the connection is unwanted, it can be given back to the original task using *t_redrq()* or closed down using *t_disrq()*.

The *t_redin()* call returns

– the task ID (TSN) of the calling task, and

– the user data that the calling task included with the redirection.

If the current task is attached for multiple TS applications, it must itself determine via suitable means the TS application to which the redirected connection belongs. Suitable means are, for example, the user data and the optional user reference to attachment of the TS application returned with *t_event()*.

```
#include <cmx.h>
int t_redin(tref, pid, opt)
int *tref;
int *pid;
union {
        struct t_optc2 optc2;
        struct t_optc3 optc3;
} *opt;
```

-> tref
Pointer to a field with the transport reference of the connection.

<- pid
Pointer to a field in which CMX(BS2000) returns the task ID of the redirecting task.

<> opt
For *opt*, specify a NULL pointer or a pointer to a union with system options.

This union is used to fetch user data that the calling task included with the redirection request (*t_redrq()*).
If *opt* = NULL is specified, CMX(BS2000) discards the user data.
If the calling task specified no user data, CMX(BS2000) returns the default values given.

The following structures are defined in <*cmx.h*>:

```
    struct t_optc2 {
->      int  t_optnr;     /* Option no. */
<-      char *t_udatap;   /* Data buffer */
<>      int  t_udatal;    /* Length of the data buffer */
    };
   struct t_optc3 {
->     int  t_optnr;    /* Option no. */
<-     char *t_udatap;  /* Data buffer */
<>     int  t_udatal;   /* Length of the data buffer */
<-     int  t_xdata;    /* Choice for expedited data */
<-     int  t_timeout;  /* Inactive time */
->     int  t_ucepid;   /* User connection reference */
    };
```

t_optnr
: Option number. Specify:

  T_OPTC2 in *t_optc2*
  T_OPTC3 in *t_optc3*

t_udatap
: Pointer to a data area in which CMX(BS2000) enters the user data received.

  Default value if NULL specified: Undefined

t_udatal
: Prior to the call, 0 or the length of the data area *t_udatap* must appear here. The area must be large enough that the received data completely fits. T_RED_SIZE, defined in <*cmx.h*>, is a suitable maximum size. CMX(BS2000) returns in this field the number of bytes received.

  Default value if NULL specified: 0

t_xdata
: In *t_xdata* the value T_NO is always returned.

t_timeout
: In *t_timeout* the value T_NO is always returned.

t_ucepid
: This field can be used to pass a freely-selectable user reference for this connection to CMX(BS2000).
  During subsequent processing this user reference can be returned to the current task by CMX(BS2000) as an option in a *t_event()* call.
  If the current task is maintaining multiple connections this mechanism enables it to associate a TS event with the appropriate connection via a user-defined attribute. The user reference constitutes an alternative the transport reference *tref*, defined by CMX(BS2000).

  Default value if NULL specified: 0

Return values

T_OK
The call was successful.
T_ERROR
Error. Query error code using *t_error()*.

Errors

If an error occurs possible error values can be queried by calling *t_error()*.
All possible error values are listed in the appendix.

See also

t_error(), t_event(), t_disrq(), t_redrq()

Note

The storage area **datap* must be assigned with read-access (**tref*) or write-access (**reason, *opt, *t_udatap*) from the program; otherwise, the program will abort with an address error. CMX(BS2000) recognizes that user data has been specified by *t_udatal > 0*, *t_udatap* = NULL is permissible!.

t_redrq
Redirect connection (redirection request)

t_redrq() redirects the specified connection to another task. The receiving task is specified by the TSN. It must be attached for the TS application to which the connection to be redirected belongs.

With *t_redrq()*, the current task may specify, in the *opt* option, user data to be passed to the receiving task when it accepts the connection. The user data can be used e.g. to inform the receiving task of the TS application to which the connection belongs.

Following the *t_redrq()* call the connection is no longer known to the calling task and the transport reference for this task is invalid. The called task must already exist and must be attached to the TS application; it receives the event T_REDIN.

The connection may not be redirected

- if T_DATASTOP or T_XDATSTOP is waiting for it, or
- while a TIDU on this connection is being fetched in piecemeal fashion with *t_datain()* (return value: n > 0).

```
#include <cmx.h>
int t_redrq(tref, pid, opt)
int *tref;
int *pid;
union {
        struct t_optc1 optc1;
        struct t_optc2 optc2;
} *opt;
```

-> tref
: Pointer to a field with the transport reference of the connection to be redirected.

-> pid
: Pointer to a field in which the TSN of the called task is to be specified.

-> opt
: For the parameter *opt*, specify the value NULL or a pointer to a union with user options. This union can be used to send information to the called task with the connection redirection. The called task receives this along with the connection redirection. If *opt* = NULL is specified, CMX(BS2000) delivers the given default values to the called task.

The following structures are defined in <*cmx.h*>:

```
    struct t_optc1 {
->      int  t_optnr;     /* Option no. */
->      char *t_udatap;   /* Data buffer */
->      int  t_udatal;    /* Length of the data buffer */
        int  t_xdata;     /* Choice for expedited data */
->      int  t_timeout;   /* Waiting period for attachment */
    };

    struct t_optc2 {
->      int  t_optnr;     /* Option no. */
->      char *t_udatap;   /* Data buffer */
->      int  t_udatal;    /* Length of the data buffer */
    };
```

t_optnr
: Option number. Specify:

T_OPTC1 in *t_optc1*
T_OPTC2 in *t_optc2*

t_udatap
: Pointer to a storage area with user data to be delivered to the receiving task.

Default value if NULL specified: Undefined

t_udatal
: Number of bytes to be transferred from the data area *t_udatap*. The maximum possible number is defined in <*cmx.h*> as T_RED_SIZE.
If *t_udatal* = 0 is specified, *t_udatap* is ignored.

Default value if NULL specified: 0

t_xdata
: This field has not yet been defined in this version. Specifications made for *t_xdata* will be ignored.

t_timeout
: This parameter is not evaluated by CMX(BS2000). The task to which the connection is to be directed must be attached to the TS application before calling *t_redrq()*.

Default value if NULL specified: T_NO

Return values

T_OK
: The call was successful.

T_ERROR
: Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_datain(), t_error(), t_event(), t_xdatin()

Note

The storage area **datap* must be assigned with read-access (**tref, *opt, *t_udatap*) from the program; otherwise, the program will abort with an address error.
CMX(BS2000) recognizes that user data has been specified by *t_udatal > 0*, *t_udatap* = NULL is permissible!.

t_strerror
Decode CMX(BS2000) error message

t_strerror() decodes CMX(BS2000) error messages passed to the task in hexadecimal form by CMX(BS2000) when *t_error()* is called.

t_strerror() returns a pointer to a static area that contains the plain text form of the CMX(BS2000) error message specified in *code*.

This text consists of error symbols, as defined in <*cmx.h*> (see below).

```
#include <cmx.h>
char *t_strerror(code)
int code;
```

-> code

For *code*, specify the representation of the error message that was passed to the task by CMX(BS2000) when *t_error()* was called.

Return values

If the call was successful, *t_strerror()* returns a pointer to a storage area with the plain text form of the CMX(BS2000) error message as a C string The plaintext consists of error symbols as defined in <*cmx.h*>. Each error symbol is preceded by \t and has the following structure:

"\t*type*\n\t*class*\n\t*value* DIAG-INF=0X*nnnnnnnn*\n"

If an undefined value is specified in *code*, *t_strerror()* returns a pointer to the text:

"\t0Xnnnnnnnn ? \n"

In case of error, *t_strerror()* returns a NULL pointer.

See also

t_error(), t_perror()

t_strreason
Decode reasons for disconnection

t_strreason() decodes reasons for disconnection passed to the task in hexadecimal form when *t_disin()* is called.

t_strreason() returns a pointer to a static area that contains the plain text form of the reason for disconnection specified in *reason*.

This text consists of the symbol for the disconnection reason, as defined in <*cmx.h*> (see below).

```
#include <cmx.h>
char *t_strreason(reason)
int reason;
```

-> reason

For *reason*, specify the representation of the disconnection reason that was passed to the task by CMX(BS2000) when *t_disin()* was called.

Return values

If the call was successful, *t_strreason()* returns a pointer to a storage area with the plain text form of the disconnection reason as a C string with the following structure:

"\t*reason*\n"

If an undefined value is specified, *t_strreason()* returns a pointer to the text:

"\t n ?\n"

In case of error, *t_strreason()* returns a NULL pointer.

Files

cmxlib.cat - Message file

See also

t_disin(), t_preason()

t_vdatain
Receive data (data indication)

t_vdatain() accepts a T_DATAIN event previously reported via *t_event()*.

By means of this call the current task receives a Transport Interface Data Unit (TIDU) of the current Transport Service Data Unit (TSDU) from the sending TS application on the specified connection.

t_vdatain() places the data of a received TIDU into a series of non-contiguous storage areas. These storage areas are described by means of the vector *vdata*.
The number of storage areas, i.e. the number of elements in *vdata*, is specified in the parameter *vcnt*.
Thus, *vcnt t_data* structures are entered in *vdata*. Each *t_data* entry describes one of the storage areas vdata[0], vdata[1],..., vdata[vcnt-1].
The data received is stored in these storage areas sequentially; each storage area is completely filled before the next one is used.
Between two TIDUs of a TSDU any other CMX(BS2000) events can occur for the same or a different connection.

The maximum length of a TIDU depends on the transport connection used. It can be queried for an established connection by means of *t_info()*.

A TIDU need not be completely full. The breakdown of a TSDU into TIDUs is purely local and does not indicate anything regarding the breakdown of the TSDU into TIDUs at the sending TS application.

t_vdatain() indicates:

– (in the *chain* parameter)
  whether a further TIDU belonging to the current TSDU exists (*chain* = T_MORE) or does not exist (*chain* = T_END).
  The individual TIDUs of a TSDU are each indicated via *t_event()* with the event T_DATAIN.

– (with the return value)
  whether the current TIDU has been completely read or not.
  If the value T_OK is returned, the TIDU has fit into the storage area provided. The current task has completely received the current TIDU.
  If a value n > 0 is returned, only a part of the TIDU has been read. n is the number of bytes of the TIDU that have not yet been read (remaining length).
  In this case *t_vdatain()* or *t_datain()* must be called repeatedly until the entire TIDU has been read. Only then can other CMX(BS2000) calls be issued again, e.g. *t_event()*.

```
#include <cmx.h>
int t_vdatain(tref, vdata, vcnt, flags)
int *tref, *vcnt, *flags;
struct t_data *vdata;
    struct t_data {
            char *t_datap; /* Datenbereich */
            int t_datal; /* Länge des Datenbereiches */
    };
```

-> tref

Pointer to a field containing the transport reference of the connection.

<> vdata

Pointer to an array of *t_data* structures for data buffers in which CMX(BS2000) enters the data of the received TIDU. The following structure is defined in <*cmx.h*>:

```
        struct t_data {
 <-         char *t_datap;  /* Data area */
 <>         int t_datal;    /* Length of the data area */
        };
```

t_datap

Pointer to a data area in which CMX(BS2000) enters data of the TIDU received.

t_datal

Prior to the call, the length of the data area *t_datap* must be entered in *t_datal* (at least 1). Following the call, CMX(BS2000) returns in this field the number of bytes entered.

-> vcnt

Number of elements in *vdata*. At least 1 and at most T_VCNT must be specified.

<- flags

Pointer to an indicator used by CMX(BS2000) to show whether there is an additional TIDU belonging to the TSDU. Possible values:

T_MORE

Another TIDU belonging to the TSDU follows. It will be indicated with a separate T_DATAIN event.

T_END

The present TIDU is the last of the TSDU.

Return values

T_OK
: The call was successful. The TIDU was completely read.

n > 0
: n bytes remain from the TIDU.

T_ERROR
: Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_datain(), t_error(), t_event(), t_info()

t_vdatarq
Send data (data request)

t_vdatarq() sends the next (or only) Transport Interface Data Unit (TIDU) of a Transport Service Data Unit (TSDU) to the receiving TS application on the specified connection.

The TIDU is provided in a series of non-contiguous storage areas.

These storage areas are defined by means of the vector *vdata*. The number of storage areas, i.e. the number of elements in *vdata*, is specified in the parameter *vcnt*.
Thus, *vcnt t_data* structures are entered in *vdata*. Each *t_data* entry describes one of the storage areas vdata[0], vdata[1],..., vdata[vcnt-1].
CMX(BS2000) takes the data sequentially from these storage areas. Each storage area is completely read before turning to the next one.

If the TSDU is longer than one TIDU, it must be transferred using several *t_vdatarq()*(or *t_datarq()*) calls in succession. Therefore in each *t_vdatarq()* call the sending task can specify in the *chain* parameter whether an additional TIDU belonging to the same TSDU follows.

The maximum length of a TIDU depends on the transport connection used. It can be queried for an established connection by means of *t_info()*.

If *t_vdatarq()* returns T_DATASTOP, the TIDU has been accepted but the flow of TIDUs on this connection has been blocked.

The flow of TIDUs can be blocked by:

- the receiving TS application,
  which can block the flow of TIDUs by calling *t_datastop()* or *t_xdatstop()*, or
- CMX(BS2000),
  if the local buffer is full.

If the flow of TIDUs is blocked, before further TIDUs can be sent you must wait, by means of *t_event()*, for the event T_DATAGO for the connection.

Successful execution of *t_vdatarq()* (T_OK) does not mean that the receiving TS application has already accepted the data. If *t_vdatarq()* fails (T_ERROR), this always indicates that a local error has been found.

```
#include <cmx.h>
int t_vdatarq(tref, vdata, vcnt, flags)
int *tref, *vcnt, *flags;
struct t_data *vdata;
    struct t_data {
            char *t_datap; /* Datenbereich */
            int t_datal; /* Länge des Datenbereiches */
    };
```

-> tref

Pointer to a field containing the transport reference of the connection.

-> vdata

Pointer to an array of *t_data* structures for data buffers from which CMX(BS2000) takes the data of the TIDU to be sent. The following structure is defined in <*cmx.h*>:

```
     struct t_data {
->       char *t_datap;  /* Data area */
->       int t_datal;    /* Length of the data area */
     };
```

t_datap

Pointer to a data area from which CMX(BS2000) takes data of the TIDU to be sent.

t_datal

For this parameter, specify the length of the data area *t_datap*. At least 1 and at most the length of a TIDU must be specified. The sum of all *t_datal* values may not exceed the maximum length of a TIDU.

-> vcnt

Number of elements in *vdata*. At least 1 and at most T_VCNT must be specified. The sum of the *t_datal* values of all *vcnt t_data* elements may not exceed the length of a TIDU.

-> flag

Pointer to an indicator used to show CMX(BS2000) whether there is an additional TIDU belonging to the TSDU. Possible values:

T_MORE

Another TIDU belonging to the TSDU follows.

T_END

The present TIDU is the last of the TSDU.

Return values

T_OK
: The call was successful; further TIDUs may be sent immediately.

T_DATASTOP
: The call was successful, but further TIDUs may not be sent until the event T_DATAGO has arrived for the specified connection.

T_ERROR
: Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_datarq(), t_datastop(), t_error(), t_event(), t_info(), t_xdatstop()

Note

> It is forbidden to continue to send data "on spec" after T_DATASTOP. Although CMX(BS2000) will not prevent this, offenders must realize that this will lead to the arrival of either too many or already invalid T_DATAGO events.
>
> The storage area **datap* must be assigned with read-access from the program; otherwise the program will abort with an address error. Null is a valid address for *t_datap*.

t_wake()
Awakening a task from t_event

The function *t_wake()* awakens the task indicated by *pid* (=TSN) from the waiting point *t_event()*. The awakened task receives the return value T_NOEVENT. *t_wake()* is used to synchronize non-CMX(BS2000) events at the CMX(BS2000) waiting point. *t_wake* can also be used to awaken another task with the same user-ID or, if called from a signal routine, its own task.

t_wake() always generates a T_NOEVENT event, even when the task to be awakened is not calling *t_event()*.

```
#include <cmx.h>
int t_wake(pid, evref)
int *pid;
int *evref;
```

-> pid
Pointer to a field with the TSN of the task to be awakened.

-> evref
The value of evref must always be NULL.

Return values

T_OK
The call was successful.

T_ERROR
Error. Query error code with *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_event(), t_error()

t_xdatgo
Release the flow of expedited data (expedited data go)

t_xdatgo() releases the blocked flow of expedited data on the specified connection. By means of this call the current task informs CMX(BS2000) that it is again ready to receive expedited data.

This call means that the current task can again receive the event T_XDATIN for the specified connection, if the event is pending.

t_xdatgo() may be called only if the exchange of expedited data was agreed when the connection was set up.

```
#include <cmx.h>
int t_xdatgo(tref)
int *tref;
```

-> tref

Pointer to a field with the transport reference of the connection on which the flow of expedited data is to be released again.

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_event(), t_error(), t_xdatstop()

Note

In CMX(BS2000), data flow control at the interface is independent of data flow control on the transport connection (due to the use of BCAM). This means *t_datastop()* - *t_datago()* is not recognized by the other TS applications until flow control on the transport connection reacts.

t_xdatin
Receive expedited data (expedited data indication)

t_xdatin() accepts a T_XDATIN event previously reported via *t_event()*. The *t_xdatin()* call must be made before the next *t_event()*.

By means of this call the current task receives an Expedited Transport Service Data Unit (ETSDU) from the sending TS application on the specified connection. The maximum length of an ETSDU depends on the transport connection used. However, it is never greater than T_EXP_SIZE bytes.

If the expedited data fits into the storage area *datap* provided, the value T_OK is returned. Otherwise, a value n > 0 is returned, where n is the number of bytes of the ETSDU that have not yet been read (remaining length). In this case, *t_xdatin()* must be called repeatedly until the entire ETSDU has been read. Only then can other CMX(BS2000) calls be issued again, e.g. *t_event()*.

```
#include <cmx.h>
int t_xdatin(tref, datap, datal)
int *tref;
char *datap;
int *datal;
```

-> tref

Pointer to a field containing the transport reference of the connection, obtained via *t_event()*.

<- datap

Pointer to a storage area in which CMX(BS2000) enters the data of the ETSDU received.

<> datal

Pointer to a field in which the length of the data area *datap* must be entered prior to the call. A value of at least 1 must be specified.
Following the call, CMX(BS2000) returns in this field the number of bytes entered.

Return values

T_OK

The call was successful. The expedited data was completely read.

n > 0

n bytes remain from the ETSDU.

T_ERROR

Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_error(), t_event()

t_xdatrq
Send expedited data (expedited data request)

t_xdatrq() sends an Expedited Transport Service Data Unit (ETSDU) with expedited data to the receiving TS application via the connection specified. The maximum length of a ETSDU depends on the transport connection used. However, it is never greater than T_EXP_SIZE bytes.

The *t_xdatrq()* call is permitted only when the exchange of expedited data was agreed when the relevant connection was set up.

ETSDUs may overtake Transport Interface Data Units (TIDUs) with normal data that had been sent earlier. It is guaranteed that ETSDUs will never arrive at the receiving TS application later than TIDUs sent after them.

If T_XDATSTOP is returned, the ETSDU has been accepted but the flow of ETSDUs and TIDUs on this connection has been blocked.

The flow of expedited data can be blocked by:

– the receiving TS application,
  which can block the flow of ETSDUs by calling *t_xdatstop()*, or

– CMX(BS2000),
  if the local buffer is full.

If the flow of ETSDUs is blocked, before further ETSDUs can be sent you must wait, by means of *t_event()*, for the event T_XDATGO or T_DATAGO for the connection.

Successful execution of *t_xdatrq()* (T_OK) does not mean that the receiving TS application has already accepted the data.
If *t_xdatrq()* fails (T_ERROR), this always indicates that a local error has been found.

```
#include <cmx.h>
int t_xdatrq(tref, datap, datal)
int *tref;
char *datap;
int *datal;
```

-> tref
  Pointer to a field with the transport reference of the connection on which the expedited data is to be sent.

-> datap
  Pointer to a storage area containing the ETSDU to be sent.

-> datal

Pointer to a field containing the number of bytes to be sent from the storage area *datap*.
Minimum value: 1
Maximum value: T_EXP_SIZE
(T_EXP_SIZE is defined in <*cmx.h*>.)

Return values

T_OK
The call was successful; further expedited data may be sent immediately.

T_XDATSTOP
The call was successful, but further ETSDUs may not be sent until the event T_XDATGO or T_DATAGO has arrived for this connection.

T_ERROR
Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_error(), t_event(), t_xdatstop()

t_xdatstop
Block the flow of expedited data (expedited data stop)

t_xdatstop() blocks the flow of both expedited and normal data on the specified connection.

More specifically, the effects of *t_xdatstop()* are:

– The current task tells CMX(BS2000) that, until further notice, it is not ready to receive normal or expedited data for this connection. However, a T_DATAIN event or a T_XDATIN event that has already been indicated must be responded to first.

– The current task no longer receives the events T_DATAIN and T_XDATIN for the specified connection. However, while the data flow is blocked it may call other CMX(BS2000) functions, e.g. to set up, close down, or redirect an additional connection.

– The send TS application can continue to send data until data flow control on the transport connection reacts.

– The sending TS application receives the return value T_XDATSTOP when it calls *t_xdatrq()* and the return value T_DATASTOP when it calls *t_datarq()*. It may not send any more normal or expedited data.

The flow of expedited data is released with *t_xdatgo()* or with *t_datago()*.

t_xdatstop() may be called only if the exchange of expedited data was agreed when the connection was set up.

```
#include <cmx.h>
int t_xdatstop(tref)
int *tref;
```

-> tref
    Pointer to a field with the transport reference of the connection.

Return values

T_OK
    The call was successful.
T_ERROR
    Error. Query error code using *t_error()*.

Errors

If an error occurs, possible error values can be queried by calling *t_error()*.
All error values are listed in the appendix.

See also

t_datago(), t_error(), t_event(), t_xdatgo(), t_xdatrq()

9 Program examples

The following two examples show an application (rcopy) for transferring text files. The application is divided into client (example 1) and server programs (example 2). In the examples, case constructs are used for the evaluation of the CMX events and the breaking up of TSDUs of any length in TIDUs (try setting tiduln at 32). In the second example, the connection - file association is only implemented using *t_ucepid*.

Example 1

```
#include <stdio.h>                    /* standard library */
#include <cmx.h>                      /* CMX library */

/*────────────────────────────────────── remote file copy ────────────────*/
/*  program rcopy()     rcopy client                                      */
/*                      copy a file to the rcopy server                   */
/*  in BS2000 set the compiler-option <parameter-prompting=YES>           */
/*────────────────────────────────────────────────────────────────────────*/

#define MAXLN 12288
struct io_rec {          /* io-buffer */
  int recnr;
  char buf[MAXLN];
  };

static struct t_opti1 opti = {T_OPTI1};
static struct t_myname   *loc_name;
static struct t_partaddr *rem_name;

static char *own_name, *ptn_name;

static char *sendar;           /* pointer to sendarea */
static int sendln = 0;         /* length of data in sendarea */

static char *file_name;
static struct io_rec *io;
static FILE *sendfile = NULL;

static int tref;
static int tiduln = 0, bytecount = 0;

static void term();
static int  send();
```

```
/*------------------------------------------- main procedure -------------------*/
/*  rcopy <file> <to-server>                                                    */
/*------------------------------------------------------------------------------*/

main(argc,argv)
 int  argc;
 char *argv[];
 {
   int reason;
   int  run=1;

  if (argc >= 2)
    {
    file_name = argv[1];
    ptn_name = argv[2];
    printf("copy %s to %s \n",file_name,ptn_name);

    /*----------------------------------------- open sendfile -------*/

    if ((sendfile = fopen(file_name,"r")) == NULL)
       term("fopen()",-1);

    /*------------------------------- get local name and open TSAP --*/

    own_name = "cmx002";
    if ((loc_name = t_getloc(own_name,NULL)) == NULL)
       term("t_getloc()",t_error());

    if (t_attach(loc_name,NULL) == T_ERROR)
       term("t_attach()",t_error());

    /*--------------------- get partner name and request connection -*/

    if ((rem_name = t_getaddr(ptn_name,NULL)) == NULL)
       term("t_getaddr()",t_error());

    if (t_conrq(&tref,rem_name,loc_name,NULL) == T_ERROR)
       term("t_conrq()",t_error());

    /*--------------------------------------- cycle until file sent -*/

     while(run)
       {
        switch (t_event(&tref,T_WAIT,NULL))
        {
         case T_CONIN: /* reject unexpected T_CONIN */
              puts("unexpected conin rejected\n");
              t_disin(&tref,NULL);
              break;

         case T_CONCF: /* confirm connection and send until T_DATASTOP */
              if (t_concf(&tref,NULL) == T_ERROR)
                 term("t_concf()",t_error());

              if (t_info(&tref,&opti) == T_ERROR)
                 term("t_info()",t_error());

              tiduln = opti.t_maxl;
              printf("connection OK, TIDULN = %d\n",tiduln);
              io = (struct io_rec *)malloc(sizeof(struct io_rec));
              io->recnr = 0;

              while (send(sendfile,tref) == T_OK);
              break;
```

```
        case T_DATAIN: /* abort transmission in case of T_DATAIN */
             t_disrq(&tref,NULL);
             fprintf("unexpected datain: transmission end\n %d bytes sendt\n",
                   bytecount);
             run = 0;
             break;

        case T_DATAGO: /* continue to send until T_DATASTOP or EOF */
             while (send(sendfile,tref) == T_OK);
             break;

        case T_DISIN: /* after EOF, server closes connection */
             t_disin(&tref,&reason,NULL);
             printf(" connection closed: %s %d bytes sent \n",
                    t_strreason(reason), bytecount);
             run = 0;
             break;

        case T_NOEVENT:  /* ignore T_NOEVENT (maybe caused by t_wake()) */
             break;

        case T_ERROR: /* terminate program */
             term("event()",t_error());
             run = 0;
             break;

        default:
             return(1);
       }
     }
    /*------------------------------------------- close TSAP  -------*/

    if (t_detach(loc_name) == T_ERROR)
       term("t_detach()",t_error());

    /*---------------------------------------- close sendfile -------*/

    if (sendfile != NULL)
       fclose(sendfile);
  }
  else puts("parameter error");

 return(0);
}

    /*-------------------------------------- procedure send ---------*/
int send(file,to)
     FILE *file;
     int  to;
{
     int dataln, ret;
     int chain;

 if (sendln == 0)    /* get record from sendfile */
    {
    sendar = (char *)io;
    sendln = 5;
    if (fgets(io->buf, MAXLN, file ) == NULL)
       io->recnr = -1;     /* set EOF-sign for server */
    else {
       io->recnr++;
       dataln = strlen(io->buf);  /* compute record-length */
       sendln += dataln;
       bytecount += dataln;
       }
    }
```

```
 do {
    if (sendln > tiduln)
       { dataln = tiduln; chain = T_MORE; }
    else
       { dataln = sendln; chain = T_END; }
    if ((ret = t_datarq(&tref, sendar, &dataln, &chain)) == T_ERROR)
       term("t_datarq",t_error());
    sendln -= dataln;
    sendar += dataln;
    } while (sendln > 0 && ret == T_OK); /* i.e. T_DATASTOP */

  if (io->recnr == -1)  /* in case of EOF stop send-cycle */
    ret = 7;

  return(ret);
}

/*----------- write error message and terminate program -----------*/

void term(msg,error)
     char *msg;
     int  error;
{
  puts("error \n");
  t_perror(msg,error);
  if (sendfile != NULL)
     fclose(sendfile);
  t_detach(loc_name);
  exit(-1);
}
```

Example 2

```
#include <stdio.h>                   /* standard library */
#include <cmx.h>                     /* CMX library */

/*--------------------------------------- remote file copy ----------------*/
/*  program rcopy()     rcopy server                                        */
/*                      copy a file from an rcopy client                    */
/*  in BS2000 set the compiler-option <parameter-prompting=YES>             */
/*-------------------------------------------------------------------------*/

#define MAXLN 12288
struct io_rec {     /* io - area */
  int recnr;
  char buf[MAXLN];
  };

typedef struct f_par {     /* record describes the receive-file */
  char name[16];           /* file-name */
  FILE *fp;                /* file-pointer */
  int  record_cnt;         /* received records */
  int  byte_cnt;           /* received bytes */
  char *rec_area;          /* pointer to actual receive area */
  int  rec_len;            /* length of received data */
  struct io_rec io;        /* io-buffer */
  } F_PAR, *F_PTR;

static struct t_opti1 opti = {aT_OPTI1};
static struct t_optc3 optc = {aT_OPTC3,NULL,0,T_YES,T_NO,0};
static struct t_opte1 opte = {aT_OPTE1,0,0,0,T_NOLIMIT,0};
static struct t_myname   *loc_name, l_name;
static struct t_partaddr rem_name;

static char *own_name, *ptn_name;

static int tref;
static int filecnt = 0;

static void term();
static int  receive();

/*--------------------------------------- main procedure ----------------*/
/*  rcopy <file> <to-server>                                            */
/*---------------------------------------------------------------------*/

main(argc,argv)
 int  argc;
 char *argv[];
 {
   int reason;
   int  run = 1;
   register F_PTR file;

  if (argc >= 1)
    {
    own_name = argv[1];
    printf("TS-application name %s \n",own_name);

    /*------------------------------- get local name and open TSAP --*/

     if ((loc_name = t_getloc(own_name,NULL)) == NULL)
       term("t_getloc",t_error());

     if (t_attach(loc_name,NULL) == T_ERROR)
       term("t_attach",t_error());
```

```
/*————————————————————————— get-event-loop —————————————————————*/

 while(run)
   {
    switch (t_event(&tref, T_WAIT, &opte))
    {
     case T_CONIN: /* create file-param and connection response */
         if (t_conin(&tref,&l_name,&rem_name,NULL) == T_ERROR)
            term("t_conin",t_error());

         if ((file = (F_PTR)memalloc(sizeof(F_PAR))) == NULL)
            term("no memory",-1);

         file->record_cnt = file->byte_cnt = file->rec_len = 0;
         file->rec_area = (char *)&file->io;
         sprintf(file->name,"rec-file.%04d",++filecnt);
         if ((file->fp = fopen(file->name,"w")) == NULL)
            term("fopen",-1);
         /*—————————— t_ucepid = address of file-par —————————————*/
         optc.t_ucepid = (int)file;

         if (t_conrs(&tref,&optc) == T_ERROR)
            term("t_conrs",t_error());

         printf("connection accepted for file %s \n", file->name);
         break;

     case T_DATAIN:  /* receive announced data */
         file = (F_PTR)opte.t_ucepid;
         if (receive(file, tref) == -1)  /* EOF-sign ? */
            {
            t_disrq(&tref,NULL);
            printf("%d bytes in %d records for file %s received\n",
                   file->byte_cnt, file->record_cnt, file->name);
            fclose(file->fp);
            memfree(file,sizeof(F_PAR));
            }
         break;

     case T_DISIN: /* connection lost */
         file = (F_PTR)opte.t_ucepid;
         t_disin(&tref,&reason,NULL);
         printf("connection lost %s %d bytes in %d records for file %s received\n",
                t_strreason(reason), file->byte_cnt, file->record_cnt, file->name);
         fclose(file->fp);
         memfree(file,sizeof(F_PAR));
         break;

     case T_NOEVENT: /* ignore T_NOEVENT (caused by t_wake() ?) */
         break;

     case T_ERROR:
         t_perror("event()",t_error());
         run = 0;
         break;

     default: /* unexpected event: terminate */
         return(1);
   }
 }
 if (t_detach(loc_name) == T_ERROR)
   term("t_detach",t_error());
}
else puts("parameter error");
```

```
 return(0);
}

 /*—— receive announced data until TSDU is complete and write record ——*/
int receive(file,from)
     register F_PTR file;
     int  from;
{
     int chain, dataln, ret;

 dataln = MAXLN - file->rec_len;
 if ((ret = t_datain(&tref,file->rec_area,&dataln,&chain)) != T_OK)
    term("t_datain",t_error());
 else {
    if (file->io.recnr == -1)  /* EOF-sign received */
       return(-1);
    file->rec_area += dataln; /* compute next receive area */
    file->rec_len += dataln;
    if (chain == T_END)
       {  /* complete TSDU received - write record to file */
       if (fputs(file->io.buf,file->fp) != 0)
          return(-1);
       file->record_cnt += 1;
       file->byte_cnt += file->rec_len - 5;
       file->rec_len = 0;
       file->rec_area = (char *)&file->io;
       }
    }

 return(ret);
}
/*—————————————————————— write error-message and terminate ————————*/
void term(msg,error)
     char *msg;
     int  error;
{
  puts("error \n");
  t_perror(msg,error);
  exit(-1);
}
```

Example 3

The following example illustrates the entries in the name service.

If the server "central" is started on the system **HARVEY** and the client "cmx002" on the computer **D016ZE01**, the following /BCAMP entries must first be made by the BCAM administrator.

on **D016ZE01**:

```
/BCMAP FU=DEF,SUBFU=LOCAL,APPL=(OSI,C'cmx002'),TSEL-I=(5,C'RCOPY')
/BCMAP FU=DEF,SUBFU=GLOBAL,APPL=(OSI,C'central'),ES=HARVEY,PTSEL-I=(8,C'RCOPYSRV')
```

on **HARVEY**:

```
/BCMAP FU=DEF,SUBFU=LOCAL,APPL=(OSI,C'central'),TSEL-I=(8,C'RCOPYSRV')
/BCMAP FU=DEF,SUBFU=GLOBAL,APPL=(OSI,C'cmx002'),ES=D016ZE01,PTSEL-I=(5,C'RCOPY')
```

The T-selectors can be predefined arbitrarily, as long as they are unique within the system. It is recommended to store the entries in a file.

10 Appendix

10.1 Comparison of CMX(BS2000) with CMX(SINIX)

Due to the different system architectures, ICMX(BS2000) differs from ICMX(SINIX). In particular, these differences are relevant to anyone who wishes to port CMX programs between SINIX and BS2000. The rules of the finite-state automaton must also be strictly adhered to in CMX(BS2000).

Parameters not evaluated in BS2000

- CMX(BS2000) does not monitor any application-specific connection limits. With *t_attach*, the parameters *t_apmode* and *t_conlim* are always set to T_ACTIVE + T_PASSIVE + T_REDIRECT and T_NOLIMIT, respectively. Incoming connection requests are always received by the oldest task of the TS application. The functionality of *t_apmode* and *t_conlim* can be replaced by suitable *t_disrq()* or *t_redrq()* calls.
- CMX(BS2000) does not monitor the inactive periods of connections. With *t_conrq()* and *t_conrs* the parameter *t_timeout* is always assumed to be T_NO.
- At redirection, the task to which the connection is to be redirected must already exist and must be attached to the TS application. With *t_redrq()* the parameter *t_timeout* is always assumed to be T_NO.

Differences in behavior of CMX(BS2000) compared to CMX(SINIX)

- With CMX(BS2000), error codes contain additional BS2000-specific diagnostic information. In the case of error types > 15, the CMX error values can be extracted with error = error & Oxff. There is no guarantee that the same error situation will return the same error value for both BS2000 and SINIX. T_WTREF in particular takes on a new meaning with CMX(BS2000):
  1. The tref is false.
  2. A call related to a connection which was in the process of being disconnected by the partner - the event T_DISIN follows (in SINIX this corresponds to T_COLLISION).

- In BS2000, a waiting state with the *t_event()* call is not automatically terminated with a signal routine. The call *t_wake()* is provided for this purpose.

Differences in the transport system

- The TNSX in BS2000 is implemented by the BCAM name service (/BCMAP command). All global names must be entered here.
- In BS2000, the maximum number of TS applications and connections can be set by a BCAM administration command within a wide range as can the waiting time for connection requests.
- In BS2000, a TS application can be closed by a BCAM administration command. This situation, which is not yet provided for SINIX, is indicated in CMX(BS2000) V1.0 via *t_event* with T_ERROR where the error value = T_CCP_END. In response to this, all program TS applications of the program should be closed and then reopened, or the program should be terminated.

10.2 CMX(BS2000) error messages

The following tables contain all possible CMX(BS2000) error messages, i.e. all error messages generated at the ICMX program interfaces. The error messages are sorted by error type and error class.

Format of CMX(BS2000) return values

Every ICMX error message is passed in the form 0x%x, where %x is a 32-bit error code. The error code is structured as follows:

```
Bit             Bit
31       12 8   0
┌─────────┬──┬────┐
│         │  │    │
└────┬────┴─┬┴──┬─┘
     │      │   │
     │      │   └──> CMX error value: The possible error values are in
     │      │                         defined in  <cmx.h>.
     │      │
     │      └──> CMX error class: Valid, if the error type is less than 15;
     │           current defined values:
     │
     │           T_CMXCLASS    0 CMX class
     │           T_DSNOT_SPEC  2 TNS class unspecified
     │           T_DSPAR_ERR   3 TNS parameter error
     │           T_DSILL_VERS  4 invalid TNS version
     │           T_DSSYS_ERR   5 TNS system error
     │           T_DSINT_ERR   6 internal TNS error
     │           T_DSMESSAGE   7 TNS information
     │
     └──> CMX error type:

          T_CMXTYPE     0 CMX error, detected by the
                          CMX library
          T_DSTEMPERR   2 temporary TNS error
          T_DSCALL_ERR  3 TNS call error
          T_DSPERM_ERR  4 permanent TNS error
          T_DSWARNING   5 TNS warning
          > 15            CMX error due to error codes from the
                          transport system.
                          Bits 8 to 31 contain diag. info, bits 0 to 7 the
                          CMX error value
```

From the CMX error value, the function *t_strerror()* produces the output string:

"\ttype\n\tclass\n\tvalue DIAG-INF = 0xnnnnnnnn \n"

The function *p_error()* outputs this string to standard error output *stderr*.

From the error values of the other CMX functions, behavior compatible with SINIX can be achieved, for example with "error = t_attach() & 0xff".

The CMX error values and their meanings are listed on the next page. The diagnostic codes (=BCAM return codes) output in DIAG-INF are listed in the tables following the CMX error values, together with their allocated values and meaning.

CMX error values

Num. value	Symbolic value	Meaning
0	T_NOERROR	No error
5	T_EIO	Temporary bottleneck or error in the transport system
14	T_EFAULT	IO area not allocated
100	T_UNSPECIFIED	Error not specified more precisely: generally an error in a system call
101	T_WSEQUENCE	Invalid call sequence
103	T_WPARAMETER	Incorrect parameter
104	T_WAPPLICATION	The application is unknown, or the task is not authorized to attach to the application, or the application has already been opened by this task.
105	T_WAPP_LIMIT	No further tasks may be attached in applications; the limit value for simultaneously active applications has been reached.
106	T_WCONN_LIMIT	Limit value for simultaneously active applications has been reached.
107	T_WTREF	Unacceptable transport reference or the transport connection has already been disconnected.
111	T_NOCCP	The transport system does not support the desired attachment or connection.
114	T_CCP_END	The transport system (BCAM) was terminated, or the application was closed by the administrator.
255	T_WLIBVERSION	No connection possible to the CMX subsystem.

Allocation of the BCAM return codes to the CMX error values

C M X error value	**DIAG - INF** S-RTC 2 1 / M-RTC 2 1	**att**	**ccf**	**cin**	**crq**	**crs**	**dgo**	**din**	**drq**	**dst**	**det**	**dsn**	**dsq**	**err**	**eve**	**inf**	**rin**	**rrq**	**vin**	**vrq**	**wak**	**xgo**	**xin**	**xrq**	**xst**
T_EFAULT	00 00 31 08							x	x										x	x					
T_EIO	00 00 00 28	x					x	x	x	x					x	x	x					x	x	x	x
	04 00 01 1C				x	x																			
	04 00 02 1C				x																				
	04 00 03 1C				x																				
	04 00 04 1C	x																							
	04 00 06 1C	x																							
	04 00 08 1C																							x	
	04 00 09 1C	x																							
	04 00 0A 1C				x																				
	04 00 0B 1C				x	x																			
	04 00 0C 1C	x			x						x		x			x									
	04 00 0D 1C	x																							
	04 00 0E 1C				x																				
	04 00 0E 1C	x																							
	04 00 13 1C	x																							
	04 00 15 1C				x	x												x							
	08 00 01 1C	x			x											x		x							
	10 00 00 1C	x																							
	00 00 01 50	x																							
	00 00 02 50	x																							
	00 00 03 50					x										x									
	00 00 01 30	x			x	x					x		x			x									
	00 00 02 30	x			x						x		x			x									
	00 00 00 10							x							x								x		
	00 00 00 2C							x							x										
	00 00 01 2C														x										
	08 00 00 10														x										

C M X error value	DIAG - INF S-RTC 2 1 / M-RTC 2 1	att	ccf	cin	crq	crs	dgo	din	drq	dst	det	dsn	dsq	err	eve	inf	rin	rrq	vin	vrq	wak	xgo	xin	xrq	xst
T_NOCCP	08 00 00 20	x																							
	0C 00 00 20	x																							
	10 00 00 20	x																							
	18 00 00 24				x																				
	1C 00 00 20	x																							
	1C 00 00 24				x																				
	1C 00 01 24				x																				
	1C 00 02 24				x																				
	1C 00 03 24				x																				
	1C 00 04 24				x																				
	1C 00 06 24				x																				
	20 00 00 20				x																				
	30 00 00 20	x																							
	38 00 00 20	x																							
	3C 00 00 20	x																							
	40 00 00 24																							x	
	40 00 04 24				x	x																			
	40 00 05 24				x	x																			
	40 00 07 24				x																				
	40 00 08 24				x																				
	04 00 01 1C					x																			
T_WAPPLICATION	00 00 04 51	x			x	x									x	x		x							
	00 00 08 51	x			x	x									x	x					x				
	00 00 0B 51																				x				
	00 00 0C 51																				x				
	00 00 0D 51															x									
	00 00 0E 51															x									
	00 00 10 51																				x				
	04 00 00 20	x			x	x	x	x	x	x	x		x		x	x	x	x				x	x	x	x
	18 00 01 20	x			x	x							x			x		x							
	40 00 00 20	x																							
T_WAPPLIMIT	28 00 00 20	x																							
T_WCONNLIMIT	24 00 00 20				x																				

C M X error value	DIAG - INF S-RTC 2 1 / M-RTC 2 1	att	ccf	cin	crq	crs	dgo	din	drq	dst	det	dsn	dsq	err	eve	inf	rin	rrq	vin	vrq	wak	xgo	xin	xrq	xst
T_WPARAMETER	- - - - - - - - -	x	x	x	x	x		x	x		x	x	x		x	x	x	x	x	x			x	x	
	00 00 00 14				x	x			x				x											x	
	00 00 02 08	x			x						x					x									
	00 00 07 08					x	x	x	x	x			x			x	x	x				x	x	x	x
	00 00 0A 08							x	x						x										
	00 00 13 08								x																
	00 00 16 08																							x	
	00 00 18 08							x	x						x										
	00 00 1F 08	x			x						x					x									
	00 00 20 08	x			x						x					x									
	00 00 21 08	x			x						x					x									
	00 00 26 08							x	x						x										
	00 00 27 08																	x			x				
	00 00 29 08				x																				
	00 00 2A 08				x																				
	00 00 2B 08				x																				
	00 00 2C 08							x							x										
	00 00 2D 08							x							x										
	00 00 2E 08															x	x								
	00 00 2F 08						x			x												x			x
	00 00 30 08															x									
	00 00 33 08	x																							
	00 00 34 08							x	x						x										
	00 00 35 08				x																				
	00 00 36 08				x																				
	00 00 37 08				x																				
	00 00 00 28																	x							
T_WSEQUENCE	- - - - - - - - -		x	x				x				x					x		x				x		
	00 00 00 3C																							x	
	00 00 00 3C																						x		
	04 00 02 24					x																			
	08 00 00 24				x																				
	0C 00 00 24				x																				
	14 00 02 24					x																			
	14 00 00 1C								x															x	
	30 00 00 24				x																				
	48 00 00 24						x	x	x	x												x	x	x	x
	5C 00 00 24																					x			x
	68 00 00 24	x			x								x												
T_WTREF	04 00 00 24					x	x	x	x	x			x		x	x	x	x				x	x	x	x

Meaning of the diagnostic codes

C M X error value	DIAG info S-RTC 2 1 | M-RTC 2 1	Meaning
T_EFAULT	00 00 31 08	User buffer not accessible
T_EIO	00 00 00 28	The call can not be executed at present (try again later)
T_EIO	04 00 01 1C	No storage area available for data buffer
T_EIO	04 00 02 1C	No free transport reference available
T_EIO	04 00 03 1C	No storage area available for ACONCB
T_EIO	04 00 04 1C	No storage area available for APPCB
T_EIO	04 00 06 1C	No storage area available for SUB-TCB
T_EIO	04 00 08 1C	BS2000 bourse mechanism overloaded
T_EIO	04 00 09 1C	No storage area available for ENACB
T_EIO	04 00 0A 1C	No storage area available for ADDRCB-P
T_EIO	04 00 0B 1C	No free CONNECTION_ID available
T_EIO	04 00 0C 1C	No storage area available for Layer 4 CB
T_EIO	04 00 0D 1C	No free APID available
T_EIO	04 00 0E 1C	No free port number available
T_EIO	04 00 10 1C	No storage area available for local event group control block available
T_EIO	04 00 11 1C	No storage area available for global event group control block available
T_EIO	04 00 13 1C	No name server entry available
T_EIO	04 00 14 1C	No storage area available for event group names
T_EIO	04 00 15 1C	No storage area available for EVOL
T_CCP_END	08 00 01 1C	BCAM shutdown announced
T_CCP_END	0C 00 01 1C	BCAM quick shutdown
T_EIO	10 00 00 1C	Global limit for the number of open TSAPs reached
T_EIO	00 00 01 50	Unknown host
T_EIO	00 00 02 50	Host not active
T_EIO	00 00 03 50	Own INTERNET_ADDRESS is invalid
T_EIO	00 00 01 30	System error when starting CONHAND processing
T_EIO	00 00 02 30	System error when waiting for CONHAND processing to end
T_EIO	00 00 00 10	No (expedited) data arrived
T_EIO	00 00 00 2C	User data has been lost
T_EIO	00 00 01 2C	Connection data has been lost
T_EIO	08 00 00 10	No telegram available
T_NOCCP	0C 00 00 20	TSAP exclusively opened by another task
T_NOCCP	10 00 00 20	TSAP already opened by this task
T_NOCCP	18 00 00 24	Partner unknown
T_NOCCP	1C 00 00 20	TSAP not active
T_NOCCP	1C 00 00 24	Partner processor unknown
T_NOCCP	1C 00 01 24	Partner processor not active
T_NOCCP	1C 00 02 24	Route(s) unknown

C M X error value	DIAG info S-RTC 2 1	DIAG info M-RTC 2 1	Meaning
T_NOCCP	1C 00 03 24		Route(s) not active
T_NOCCP	1C 00 04 24		Partner IP address unknown
T_NOCCP	1C 00 06 24		Connection request to broadcast address
T_NOCCP	20 00 00 20		*t_conrq* not permitted for TSAP
T_NOCCP	2C 00 00 20		TSAP password invalid
T_NOCCP	30 00 00 20		TSAP could not be reopened
T_NOCCP	38 00 00 20		TSAP would have been reopened
T_NOCCP	3C 00 00 20		No access to TSAP via CMX
T_NOCCP	40 00 00 24		The specified connection does not permit the sending of expedited data
T_NOCCP	40 00 04 24		Connection data not permitted
T_NOCCP	40 00 05 24		Required interface functionality is not supported
T_NOCCP	40 00 05 24		Required interface functionality is not supported
T_NOCCP	40 00 07 24		The partner' s functionality is not compatible
T_NOCCP	40 00 08 24		Level 4 address not available
T_NOCCP	04 00 01 1C		No storage area available for the data buffer
T_WAPPLICATION	00 00 01 51		Event group of another task already open and shareable
T_WAPPLICATION	00 00 02 51		Event group already opened by this task
T_WAPPLICATION	00 00 03 51		Event group already exclusively opened by another task
T_WAPPLICATION	00 00 04 51		Task not attached to the event group
T_WAPPLICATION	00 00 06 51		Standard event group not exclusively opened
T_WAPPLICATION	00 00 08 51		Standard event group not opened
T_WAPPLICATION	00 00 0B 51		No local task event group specified
T_WAPPLICATION	00 00 0C 51		Caller does not have the same user ID as the owner of the event group
T_WAPPLICATION	00 00 0D 51		The connection-specific events are not reported to the event group
T_WAPPLICATION	00 00 0E 51		The TSAP-specific events are not reported to the event group
T_WAPPLICATION	00 00 10 51		The user ID of the owner of the event group could not be determined
T_WAPPLICATION	00 00 12 51		EVENT_ID of the event group is invalid
T_WAPPLICATION	00 00 13 51		The event group cannot be closed as it is still in use
T_WAPPLICATION	04 00 00 20		TSAP not opened by this task
T_WAPPLICATION	18 00 01 20		TSAP has just been forced to close by the BCAM administrator
T_WAPPLICATION	40 00 00 20		Privileges for opening the TSAP are not available
T_WAPPLIMIT	28 00 00 20		Local task limits for the number of open TSAPs has been reached
T_WCONNLIMIT	24 00 00 20		No further connections allowed for this TSAP
T_WPARAMETER	00 00 00 14		User data length too high
T_WPARAMETER	00 00 00 14		Length of expedited or connection data too high
T_WPARAMETER	00 00 02 08		Incorrect syntax of the NEA TSAP names

C M X error value	**DIAG info** S-RTC 2 1 / M-RTC 2 1	**Meaning**
T_WPARAMETER	00 00 03 08	TSAPOPEN_ID not specified
T_WPARAMETER	00 00 07 08	CONNECTION_ID not specified
T_WPARAMETER	00 00 0A 08	LENGTH_OF_USER_BUFFER invalid
T_WPARAMETER	00 00 13 08	User buffer length = 0
T_WPARAMETER	00 00 16 08	Expedited data length = 0
T_WPARAMETER	00 00 18 08	NUMBER_OF_USER_BUFFER not specified
T_WPARAMETER	00 00 1B 08	Invalid number of route names
T_WPARAMETER	00 00 1F 08	NEA TSAP name not specified
T_WPARAMETER	00 00 20 08	No ISO TSAP name specified
T_WPARAMETER	00 00 21 08	Length of TSAP name not specified
T_WPARAMETER	00 00 26 08	LENGTH_OF_USER_BUFFER_2 invalid
T_WPARAMETER	00 00 27 08	Invalid WAKE_TSN
T_WPARAMETER	00 00 29 08	Partner TSAP name not specified
T_WPARAMETER	00 00 2A 08	No ISO partner TSAP name specified
T_WPARAMETER	00 00 2B 08	Length of partner TSAP name not specified
T_WPARAMETER	00 00 2C 08	No expedited data available
T_WPARAMETER	00 00 2D 08	No standard data available
T_WPARAMETER	00 00 2E 08	TYPE_OF_INFORMATION invalid
T_WPARAMETER	00 00 2F 08	TYPE_OF_TRANSFER_INDICATION invalid
T_WPARAMETER	00 00 30 08	Invalid socket host name
T_WPARAMETER	00 00 33 08	Specified port number is in use
T_WPARAMETER	00 00 34 08	LENGTH_OF_USER_BUFFER_1 invalid
T_WPARAMETER	00 00 35 08	The address information specified contradicts that specified by the /BCMAP command
T_WPARAMETER	00 00 36 08	LEVEL_3_CONNECTION_USER_DATA incorrect
T_WPARAMETER	00 00 37 08	End system name (parameter specified or defined by the /BCMAP command) contradicts the IP address specified
		t_redrq cannot currently be executed (try again later)
T_WPARAMETER	00 00 00 28	The specified connection does not permit expedited data to be sent or received
T_WSEQUENCE	00 00 00 3C	The use of expedited data is not permitted
T_WSEQUENCE	04 00 02 24	Connection already exists
T_WSEQUENCE	08 00 00 24	Connection being set up
T_WSEQUENCE	0C 00 00 24	No connection request pending
T_WSEQUENCE	14 00 02 24	Wait for DATA_GO_INDICATION
T_WSEQUENCE	14 00 00 1C	Wait for EXPDATA_GO_INDICATION
T_WSEQUENCE	14 00 00 1C	TSAP not authorized to set up connections
T_WSEQUENCE	30 00 00 24	Connection is not in the data transfer phase (not fully set up)
T_WSEQUENCE	48 00 00 24	PORT number already in use
T_WSEQUENCE	60 00 00 24	Connection has already been closed down
T_WSEQUENCE	64 00 00 24	The specified NEA address is currently being used by
T_WSEQUENCE	68 00 00 24	another TSAP
T_WTREF	04 00 00 24	Invalid CONNECTION_ID

10.3 List of reasons for disconnection

The reasons for disconnection passed by CMX(BS2000) in *reason* following the calls *t_disin()* and *x_disin()* are described below. The symbolic values specified here are numerically defined in <*cmx.h*>. "Local transport system" stands for the transport system in the system of the current task, while "partner transport system" stands for the transport system in the system of the connection partner of the current task.

Reasons given by CMX(BS2000):

Num. value	Symbolic value	Meaning
0	T_USER	Disconnection by the communication partner; possibly also due to a user error on the part of the communication partner
1	T_RTIMEOUT	Local disconnection by CMX due to inactivity on the connection as specified by the parameter t_timeout.
2	T_RADMIN	Local disconnection by CMX due to deactivation of the transport system by administration
3	T_RCCPEND	Local disconnection by CMX due to transport system failure

Reasons given by the partner transport system:

Num. value	Symbolic value	Meaning
256	T_RUNKNOWN	Disconnection by the partner or the transport system; no reason specified.
257	T_RSAPCONGEST	Disconnection by the partner transport system due to a TSAP-specific bottleneck.
258	T_RSAPNOTATT	Disconnection by the partner transport system because the TSAP addressed is not attached there.
259	T_RUNSAP	Disconnection by the partner transport system because the TSAP addressed is not known there.
261	T_RPERMLOST	Disconnection by network administration or by the administration of the partner transport system
262	T_RSYSERR	Error in network.
385	T_RCONGEST	Disconnection by the partner CCP due to resource bottleneck.
386	T_RCONNFAIL	Disconnection by the partner transport system due to failure in connection setup. Connection setup may fail e.g. because user data is too long or expedited data is not permitted.
387	T_RDUPREF	Disconnection by the partner transport system because asecond connection reference was assigned for an NSAP pair (system error).
388	T_RMISREF	Disconnection by the partner transport system due to a connection reference that could not be assigned (system error).
389	T_RPROTERR	Disconnection by the partner transport system due to a protocol error (system error).
391	T_RREFOFLOW	Disconnection by the partner transport system due to connection reference overflow.
392	T_RNOCONN	Establishment of the network connection rejected by the partner transport system.
394	T_RINLNG	Disconnection by the partner transport system due to incorrect header or parameter length (system error).

Reasons given by the local transport system:

Num. value	Symbolic value	Meaning
448	T_RLCONGEST	Disconnection by the local transport system due to resource bottleneck.
449	T_RLNOQOS	Disconnection by the local transport system because quality of service can no longer be provided.
451	T_RILLPWD	Invalid (connection) password.
452	T_RNETACC	Network access refused.
464	T_RLPROTERR	Disconnection by the local transport system due to a transport protocol error (system error).
465	T_RLINTIDU	Disconnection by the local transport system because an interface data unit (TIDU) over the maximum permissible length was received.
466	T_RLNORMFLOW	Disconnection by the local transport system due to violation of flow control rules for normal data (system error).
467	T_RLEXFLOW	Disconnection by the local transport system due to violation of the flow control rules for expedited data (system error).
468	T_RLINSAPID	Disconnection by the local transport system because it received an invalid TSAP ID (system error).
469	T_RLINCEPID	Disconnection by the local transport system because it received an invalid TCEP ID (system error).
470	T_RLINPAR	Disconnection by the local transport system due to an illegal parameter value, e.g. user data too long or expedited data not permitted.
480	T_RLNOPERM	Connection setup prevented by the administration of the local transport system.
481	T_RLPERMLOST	Disconnection by the administration of the local transport system.
482	T_RLNOCONN	Connection could not be set up by the local transport system because no network connection available.
483	T_RLCONNLOST	Disconnection by the local transport system transport due to loss of the network connection.
484	T_RLNORESP	Connection could not be set up by the local transport system because the partner does not respond to CONRQ.
485	T_RLIDLETRAF	Disconnection by the local transport system due to loss of the connection (Idle Traffic Timeout).
486	T_RLRESYNC	Disconnection by the local transport system because resynchronization was unsuccessful (more than 10 repetitions).
487	T_RLEXLOST	Disconnection by the local transport system because the expedited data channel is defective (more than 3 repetitions).

Glossary

Active partner

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

Address

see *TRANSDATA address* and *TRANSPORT ADDRESS*.

Application

see *TS application*.

ASCII

International character set for DP systems based on a 7 bit code (ISO 7 bit code).

CCITT

Organization of public network operators and PTTs based in Geneva.

Communication method

An access method that uses the transport services defined in the *OSI Reference Model*.

Communication partner

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

Connection, virtual

An association between two *communication partners* which allows them to exchange data with each other.

Data unit

The set of characters that can be sent in one go with the t_datarq() call or received in one go with t_datain().

DCAM application

A *TS application* in BS2000 which uses the DCAM access method.

EBCDIC

EBCDIC is an extended 8-bit version of BCD code which is used on BS2000 mainframes, TRANSDATA communication computers, and IBM-compatible systems.

ETSDU

Expedited data unit.

GLOBAL NAME

Name of a *TS application* which uniquely identifies it in the network. The *TS application* is registered in the *TS directory* under its GLOBAL NAME.

ISO Reference Model

Model for communication in open systems. It is described in ISO standard 7498 and comprises 7 layers.

KOGS

Special language used for describing network configurations.

LOCAL NAME

Property of a *TS application* in the *TS directory* associated with the *GLOBAL NAME*. The LOCAL NAME must be specified when attaching to CMX(BS2000).

Message

A logically related set of data which is to be sent to *communication partner*.

Partner

see *communication partner*.

Passive partner

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

Processor

Network-wide addressable entity in a host or communication system which provides the functionality of the transport service.

Processor name

A part of the *TRANSDATA address*. The processor name is specified in the form: processor number/region number

TIDU

Data unit

TRANSPORT ADDRESS

Property of a *TS application* in the *TS directory* associated with the *GLOBAL NAME*. The TRANSPORT ADDRESS must be specified when setting up a connection to a communication partner. The value of this property is the transport address required by CMX(BS2000).

Transport Layer

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

Transport Name Service in SINIX and SINIX-ODT

Service in CMX(SINIX) for managing transport system-specific properties of *TS applications*.

Transport reference

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

Transport system

The bottom four layers of the *OSI Reference Model*.

TSAP

Used by a *TS application* to access the transport system.

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.

TSDU

Message.

Abbreviations

ASCII	American Standard Code for Information Interchange
CMX	Communication Method in SINIX
BCAM	Basic Communication Access Method im BS2000
CCITT	Comite Consultatif International Telegraphique et Telephonique
DCAM	Data Communication Access Method
EBCDIC	Extended Binary Coded Decimal Interchange Code
EMDS	Emulation display terminal
EOF	End of File
EOS	End of String
ETSDU	Expedited Transport Service Data Unit
FT	File Transfer
ICMX	Interface to Communication Method in SINIX and SINIX Open Desktop
ISO	International Organization for Standardization
KOGS	Configuration-oriented generator language
LAN	Local Area Network
NEA	Network architecture for TRANSDATA systems
OSI	Open Systems Interconnection
PDN	Program system for telecommunication and network control

TCEP	Transport Connection Endpoint
TIDU	Transport Interface Data Unit
TS	Transport Service
TSAP	Transport Service Access Point
TSDU	Transport Service Data Unit
WAN	Wide Area Network

Related publications

In Germany, ISO standards can be obtained from:

DIN Deutsches Institut für Normung
Burggrafenstr. 4-10, Postfach 1107
D - 1000 Berlin 30

ISO 8072-1986
Information processing systems - Open Systems
Interconnection - Transport service definition

ISO 8073-1986
Information processing systems - Open Systems
Interconnection - Connection oriented transport protocol specification

ISO 7498-1984
Information processing systems - Open Systems

Interconnection - Basic Reference Model

Please apply to your local office for ordering the manuals.

Index

L

M

N

O

P

R

S

T

Contents

CMX (BS2000) V1.0A

Communication Method in BS2000

Target group

Programmers of transport service (TS) applications

Contents

CMX (BS2000) offers application programs a uniform interface to the transport services. You can use CMX (BS2000) to create application programs which can communicate with other applications independently of the transport system.

Edition: September 1992

File: CMX.PDF

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *…@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at
http://ts.fujitsu.com/...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *…@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.