1 Einleitung

1.1 Kurzbeschreibung des Produkts CMX(BS2000)

Das Transportzugriffssystem CMX(BS2000) ist eines der Produkte im BS2000 (DCAM, CMX(BS2000), SOCKETS(BS2000)), die eine Schnittstelle zu dem Transportsystem BCAM (**B**asic **C**ommunication **A**ccess **M**ethod) bieten. Die Schnittstelle steht auch in den Betriebssystemen SINIX und MSDOS zur Verfügung. Mit CMX(BS2000) lassen sich Anwendungsprogramme erstellen, die unabhängig vom Transportsystem mit anderen Anwendungen kommunizieren können.

1.2 Zielgruppen des Handbuchs

Das Manual richtet sich an den Programmierer von Transport-Service-Anwendungen, abgekürzt TS-Anwendungen. Diese TS-Anwendungen dienen zur Kommunikation und bestehen aus Anwendungsprogrammen, die in C implementiert sind.

Es wird die Beherrschung der Programmiersprache C und des C-Entwicklungssystems erwartet. Ferner ist es für das Verständnis hilfreich, Prinzipien und Methoden der Datenfernverarbeitung zu kennen, insbesondere das ISO-Referenzmodell, wie in DIN ISO 7498 normiert.

1.3 Konzept der Beschreibung von CMX(BS2000)

Das vorliegende Handbuch CMX(BS2000) beschreibt die Programmschnittstellen von CMX(BS2000), d.h. alle Werkzeuge, die Sie benötigen, um selbst TS-Anwendungen zu entwickeln.
Hilfsmittel zur Diagnose finden Sie im Anhang.

Aufbau dieses Handbuchs

Das vorliegende Handbuch ist in zwei Teile gegliedert:

Der 1. Teil ist zum Kennenlernen von CMX(BS2000) gedacht und soll dem Einsteiger helfen, TS-Anwendungen zu erstellen.
Er beschreibt die Abbildung einer TS-Anwendung auf das Taskkonzept Ihres Systems und die Zuordnung der Transportverbindungen zu den Tasks der TS-Anwendung.
Es wird die Strukturierung einer TS-Anwendung in drei Kommunikationsphasen beschrieben und aufgezeigt, wie die Funktionen der Programmschnittstellen innerhalb dieser Phasen angewendet werden. Es wird erläutert, wie Sie im Fehlerfall Diagnoseinformationen von CMX(BS2000) abfragen können. Zu den einzelnen Programmierschritten finden Sie Programmfragmente als Beispiele.

Der 2. Teil umfaßt das Kapitel "Die Programmschnittstelle ICMX". In diesem Kapitel werden die CMX(BS2000)-Programmschnittstelle sowie jeder einzelne Funktionsaufruf dieser Schnittstelle und seine Parameter im Detail beschrieben. Die Beschreibung erfolgt in alphabetischer Reihenfolge. Am Anfang des Kapitels finden Sie eine Zusammenfassung aller Informationen, die Sie zur Anwendung der Funktionen benötigen.

Bei der Beschreibung der Programmschnittstelle werden alle Möglichkeiten der Anbindung eines Rechners ans Netz (LAN und WAN) berücksichtigt.

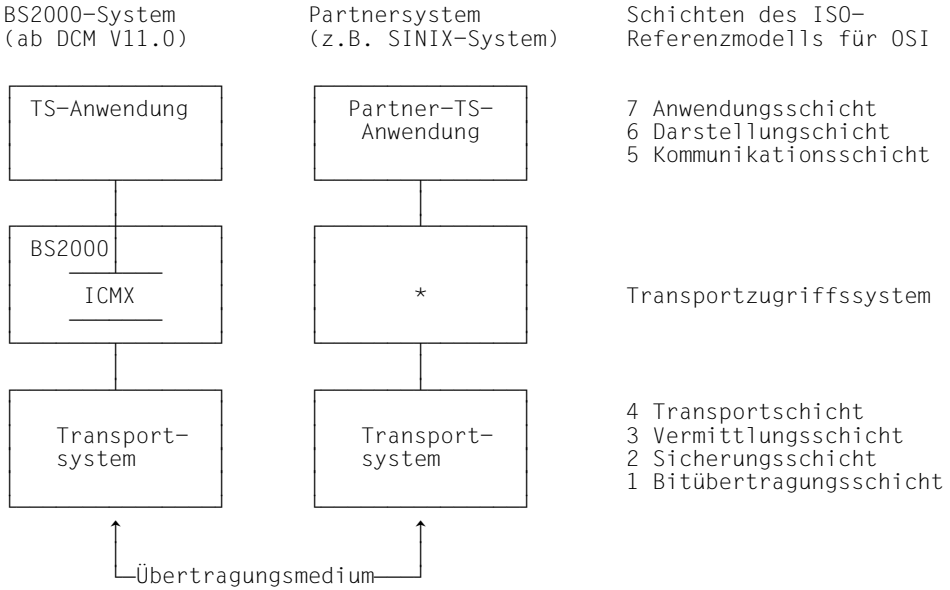
Die Programmschnittstelle ist unabhänig vom jeweiligen Betriebssystem beschrieben.

2 Das Transportzugriffssystem CMX(BS2000)

Eine Anwendung, die Daten mit einer Anwendung in einem anderen Endsystem austauschen will, benötigt die Dienste eines Transportsystems. Das Transportsystem übernimmt alle Aufgaben, die zur Vermittlung der Verbindung und zum Transport der Daten über die physikalischen Medien (Leitung, Rechner) benötigt werden. Anwendungen, die die Dienste eines Transportsystems nutzen, nennt man TS-Anwendungen.

Eine TS-Anwendung sollte über verschiedene Kopplungsmechanismen Verbindungen aufbauen und Daten austauschen können. Eine TS-Anwendung sollte also möglichst unabhängig von den jeweils zugrundeliegenden Kopplungsmechanismen sein. Diese unterscheiden sich z.B. in der Größe der Dateneinheit, die übertragen werden kann, im Format der zu übergebenden Transportadresse der Partneranwendung und im Format der Adresse der TS-Anwendung im lokalen System.
CMX(BS2000) bietet den TS-Anwendungen aus diesem Grund eine einheitliche Schnittstelle an, die Programmschnittstelle ICMX. An dieser Schnittstelle stehen den TS-Anwendungen die Dienste der Transportsysteme, die den Regeln des ISO-Referenzmodells für offene Systeme entsprechen, zur Verfügung. CMX(BS2000) ist also ein Transportzugriffssystem.

2.1 Kommunikation zwischen TS-Anwendungen



Das Transportzugriffssystem CMX(BS2000)

Eine TS-Anwendung, die die Funktionen von CMX(BS2000) nutzt, kann in einheitlicher Weise mit folgenden TS-Anwendungen kommunizieren:

– anderen TS-Anwendungen in demselben Rechner (lokale Kommunikation)

– TS-Anwendungen in SINIX- oder SINIX-ODT-Rechnern, die die Funktionen des Transportzugriffssystems CMX(L) nutzen

– TS-Anwendungen in Verarbeitungsrechnern (VAR) mit BS2000, die die Funktionen des Transportzugriffssystems DCAM, CMX(BS2000) oder von UTM nutzen

– TS-Anwendungen in Kommunikationsrechnern (KR) mit PDN, die die Funktionen des Transportzugriffssystems CAM nutzen

– TS-Anwendungen in Fremdsystemen, sofern sie die im OSI-Modell aufgeschriebenen Normen erfüllen (z.B. ICP/IC mit RFC 1006).

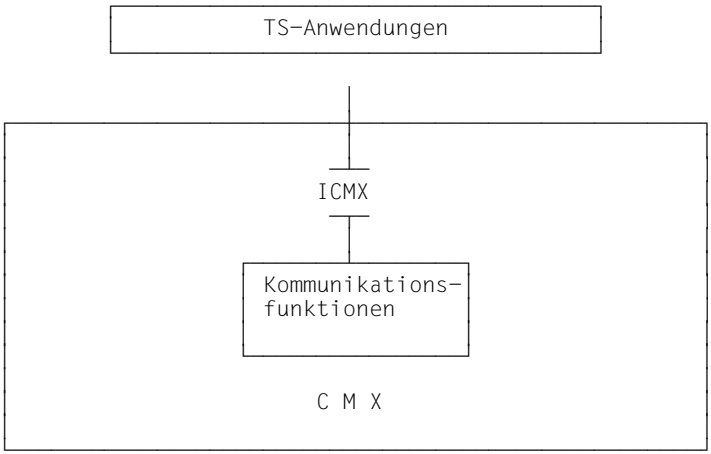
Für den Programmierer bedeutet die einheitliche Programmschnittstelle ICMX, daß er unabhängig von den spezifischen Eigenschaften der Datenübermittlung TS-Anwendungen entwickeln kann. Er programmiert für die Kommunikation nur die Funktionen von ICMX, die dazu dienen:

– seine TS-Anwendung bei CMX(BS2000) anzumelden,
– Transportverbindungen zu Partneranwendungen aufzubauen,
– Daten zu senden und zu empfangen,
– den Datenfluß zu regeln,
– Transportverbindungen abzubauen,
– seine TS-Anwendungen bei CMX(BS2000) abzumelden.

TS-Anwendungen, die die Funktionen der Schnittstellen von CMX(BS2000) nutzen, werden in der folgenden Beschreibung auch CMX(BS2000)-Anwendungen genannt. Dieser Ausdruck wird immer dann verwendet, wenn es nötig ist, über CMX(BS2000) laufende TS-Anwendungen und andere TS-Anwendungen zu unterscheiden.

2.2 Die Programmschnittstelle von CMX(BS2000) - eine Übersicht

CMX(BS2000) bietet dem Programmierer von TS-Anwendungen Funktionen zur verbindungsorientierten Kommunikation an. Diese Funktionen umfassen lokale Dienste, die Verbindungsverwaltung und den Datenaustausch. Sie stehen über die Programmschnittstelle ICMX zur Verfügung.



CMX(BS2000)-Progammschnittstelle

Die Programmschnittstelle von CMX(BS2000) ist eine Bibliotheksschnittstelle. Das heißt, die Funktionen von CMX(BS2000) stehen in Form von einem Anschlußmodul (YDCMXLNK) und einem Bibliotheksgroßmodul (YDCMXLIB) zur Verfügung. Das Anschlußmodul befindet sich zusammen mit der Include-Datei *cmx.h* in der Bibliothek SYSLIB.CMX.010. Das Bibliothekgroßmodul wird in der Bibliothek SYSLNK.CMX.010 ausgeliefert und als Subsystem CMX-TU installiert.

2.2.1 CMX(BS2000)-Funktionen zur Kommunikation (ICMX)

Die Programmschnittstelle von CMX(BS2000) (ICMX) umfaßt alle Funktionen, die eine TS-Anwendung zur Kommunikation benötigt.

Folgende Funktionsgruppen stehen an der Programmschnittstelle zur Verfügung:

Funktionen zum An- und Abmelden bei CMX(BS2000)

Bei der Anmeldung (attach) übergibt die TS-Anwendung ihre eigene Adresse innerhalb des lokalen Systems, ihren LOKALEN NAMEN, an CMX(BS2000). Erst dann ist die TS-Anwendung adressierbar. Nach der Kommunikation muß sich die TS-Anwendung bei CMX(BS2000) abmelden (detach).

Funktionen zum Aufbau einer Verbindung

Dazu gehören folgende Funktionen:

– aktiver Verbindungsaufbau
  Die beiden Funktionen dieser Gruppe dienen dazu, die Verbindung bei der fernen TS-Anwendung anzufordern (connection request) und die Verbindung nach der positiven Antwort der fernen TS-Anwendung herzustellen (connection confirmation).

– passive Verbindungsannahme
  Die beiden Funktionen dieser Gruppe dienen dazu, den Wunsch zum Verbindungsaufbau von einer fernen TS-Anwendung entgegenzunehmen (connection indication) und diese Anfrage zu beantworten (connection response).

Funktionen zum Abbau einer Verbindung

Die beiden Funktionen dieser Gruppe dienen dazu, die Verbindung abzubauen (disconnection request) bzw. den Verbindungsabbau entgegenzunehmen (disconnection indication).

Funktionen zum Umlenken einer Verbindung

Innerhalb einer TS-Anwendung kann eine Verbindung an eine andere Task derselben TS-Anwendung weitergegeben (umgelenkt) werden. Die beiden Funktionen dieser Gruppe dienen dazu, eine Verbindung umzulenken (redirect request) und eine Verbindung von einer anderen Task entgegenzunehmen (redirect indication).

Funktionen zum Austausch von Daten

Mit diesen Funktionen können Sie wie folgt Daten austauschen:

– Normaldaten senden (data request) und empfangen (data indication).
– Vorrangdaten senden (expedited data request) und empfangen (expedited data indication).
  Vorrangdaten sind kleine Datenmengen, die mit "Vorrang" vor dem Hauptstrom der Daten zu einem Kommunikationspartner übertragen werden. Diese Funktionen sind optional.

Funktionen zur Flußregelung

Wenn Sie gerade keine Daten empfangen wollen oder können, teilen Sie CMX(BS2000) dies mit. CMX(BS2000) zeigt dann keinen Datenempfang mehr an. Dem Kommunikationspartner wird dies (in der Regel) mitgeteilt, er darf dann nichts mehr senden, bis Sie den Datenfluß wieder freigeben. Der Datenfluß kann getrennt für Daten und Vorrangdaten geregelt werden (datastop,datago,xdatstop,xdatgo).

Funktionen zur Abfrage von Informationen

Mit dieser Gruppe von Funktionen können Sie folgende Informationen einholen:

– ein Ereignis (event) abwarten oder abholen.
  Ein Ereignis ist z.B. der Abbau einer Verbindung durch den Kommunikationspartner.
– Fehler (error) abfragen.
– Information (info) über CMX(BS2000)-Parameter abfragen.
– LOKALE und GLOBALE NAMEN, TRANSPORTADRESSEN abfragen (get local name, get name, get address).

Funktion zum Synchronisieren anderer Ereignisse

Mit dieser Funktion können Sie eine (andere oder die eigene) Task aus dem Wartezustand wecken (*wake*).

In den Kapiteln "Ereignisverarbeitung und Fehlerbehandlung", "An-/Abmelden bei CMX(BS2000)", "Verbindungen verwalten" und "Daten übertragen" wird die Verwendung der Funktionen in den Programmen einer TS-Anwendung erläutert. Die Funktionsaufrufe sind im Kapitel "Die Programmschnittstelle ICMX" detailliert beschrieben.

2.2.2 Optionen des Systems und des Benutzers

Der Funktionsumfang der Programmschnittstellen von CMX(BS2000) besteht aus obligatorischen und optionalen Funktionen mit obligatorischen und optionalen Parametern.
Für die Kommunikation mit Partnern über CMX(BS2000) stehen bei allen Transportverbindungen stets die obligatorischen Funktionen mit den obligatorischen Parametern zur Verfügung.
Abhängig von der verwendeten Anbindung ans Netz, stehen auch optionale Funktionen zur Verfügung, sowie optionale Parameter in den obligatorischen Funktionen.

Die Optionen sind die folgenden:

Option	**optionale Funktion**	**optionaler Parameter**	**s/b**
Benutzerdaten beim Verbindungsaufbau	n	j	s/b
Benutzerdaten beim Verbindungsabbau	n	j	s/b
Vorrangdaten	j	j	s/b
Überwachung der Inaktivzeit *)	n	j	s/b
Verbindungslimit, Aktiv/Passiv-Modus *)	n	j	b
Benutzerreferenz der Anmeldung	n	j	b
Benutzerreferenz der Verbindung	n	j	b
Zeitschranke bei synchroner Ereignisverarbeitung (± 60 Sek.)	n	j	b
Wartezeit bei der Verbindungsumlenkung *)	n	j	b
n/j = nein/ja * = nicht im CMX(BS2000) s = Systemoption b = Benutzeroption			

Tabelle 1: Tabelle Optionen von CMX(BS2000)

Die Systemoptionen richten sich nach dem Funktionsumfang der genutzten Transportverbindungen. Wenn man Optionen verwendet, die das Transportsystem oder die Kommunikationsschnittstelle der Partneranwendung nicht bietet, kommt der Verbindungsaufbau nicht zustande, oder man erhält eine Verbindungsabbauanzeige von CMX(BS2000). Bei Bereitstellung geeigneter Transportsysteme garantiert CMX(BS2000) den einwandfreien Ablauf Ihrer CMX(BS2000)-Anwendung.

Für eine einwandfreie Kommunikation müssen auch die Benutzeroptionen stimmen, d.h. die Partner ein gemeinsames Verständnis von deren Verwendung haben.

Das bedeutet, CMX(BS2000) gleicht *nicht* die Differenz zwischen der in der TS-Anwendung erwarteten und der von den Transportsystemen tatsächlich gebotenen Funktionalität aus. Dies gilt insbesondere für die obigen Systemoptionen.

3 TS-Anwendungen

Dieses Kapitel skizziert die Charakteristika von TS-Anwendungen, die die Funktionen der Programmschnittstellen von CMX(BS2000) nutzen.

In den Abschnitten dieses Kapitels werden die folgenden Punkte betrachtet:

– Namen und Eigenschaften einer TS-Anwendung
  Jede TS-Anwendung hat einen GLOBALEN NAMEN, unter dem sie im Netz eindeutig identifizierbar ist. Zur Kommunikation mit anderen TS-Anwendungen im Netz muß eine TS-Anwendung adressierbar sein. Deshalb wird einer TS-Anwendung neben anderen Eigenschaften auch die Eigenschaft TRANSPORTADRESSE bzw. LOKALER NAME zugeordnet.

– Struktur einer TS-Anwendung
  Eine TS-Anwendung ist ein C-Programm oder ein System von C-Programmen, das die CMX(BS2000)-Funktionen aufruft.
  Es wird beschrieben, was Sie beachten müssen, wenn Sie solche TS-Anwendungsprogramme erstellen, wie diese C-Programme übersetzt werden und welche Bibliotheken zum Quellcode gebunden werden müssen.

– Zusammenhang zwischen TS-Anwendung, Tasks und Verbindung
  Es wird beschrieben, wie sich eine TS-Anwendung auf das Taskkonzept des Systems abbilden läßt, und die Zuordnung Task $\longleftrightarrow$ Verbindung veranschaulicht.

3.1 Namen und Eigenschaften von TS-Anwendungen

Jede TS-Anwendung besitzt einen GLOBALEN NAMEN. Dieser Name identifiziert die TS-Anwendung eindeutig im Netz. D.h. verschiedene TS-Anwendungen haben verschiedene GLOBALE NAMEN. Der GLOBALE NAME gibt an, um welche TS-Anwendung es sich handelt.

Die GLOBALEN NAMEN aller TS-Anwendungen im lokalen System und aller TS-Anwendungen in fernen Systemen, mit denen die lokalen TS-Anwendungen kommunizieren wollen, stehen in einem Adreß- und Namensverzeichnis des lokalen Transportsystems.

Das Adreß- und Namensverzeichnis ist im BS2000 durch die Administrationsfunktion BCAM-Mapping realisiert (siehe /BCMAP-Kommando im Handbuch "BS2000 Systembedienung"). Die Generierung der Einträge im Transport Name Service ist Aufgabe des System- bzw. Netzadministrators und daher nicht Gegenstand dieses Handbuchs. Zur Generierung der Einträge muß der Entwickler einer TS-Anwendung die Namen der eigenen TS-Anwendung und die aller erreichbaren Partner sowie die Art der Kopplung dem Administrator bekanntgeben.

3.1.1 Der GLOBALE NAME einer TS-Anwendung

Der GLOBALE NAME einer TS-Anwendung ist ein hierarchisch strukturierter Name. Er besteht aus maximal 5 Teilen: Namensteil[1], Namensteil[2]... Namensteil[5]. Von diesen ist Namensteil[1] in der Hierarchie der höchste, Namensteil[5] der niedrigste. In einem GLOBALEN NAMEN müssen nicht alle Hierarchiestufen vorhanden sein. Es können auch Namensteile übersprungen werden. Ein GLOBALER NAME kann auch nur aus einem Namensteil einer beliebigen Hierarchiestufe bestehen.

Beispiele für GLOBALE NAMEN sind:

	Nt[1]	**Nt[2]**	**Nt[3]**	**Nt[4]**	**Nt[5]**
GLOBALER NAME 1	D	Siemens AG	Mch-P	DF1	G._Meier
GLOBALER NAME 2		Abteilung A	Reg18	Proz. 1	$DIALOG
GLOBALER NAME 3	49	089	636		47658

Nt = Namensteil

Die GLOBALEN NAMEN werden in den C-Prozeduren analog zu SINIX (Namensteile mit Punkt "." getrennt) geschrieben (z.B. *franz.xyz.1*). Beim Eintragen in den Transport Name Service (*/BCMAP*-Kommando) dagegen müssen dort die Namensteile mit X’00’ getrennt werden.

Beispiel

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

Dem Namen "franz.xyz.1" (=X'86998195A94BA7A8A94BF1') wird mit folgendem /BCMAP-Kommando der T-Selektor "TEST001" für den LOKALEN NAMEN zugeordnet:

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

(siehe hierzu auch Kapitel "Programmbeispiele").

3.1.2 Die Eigenschaften LOKALER NAME und TRANSPORTADRESSE

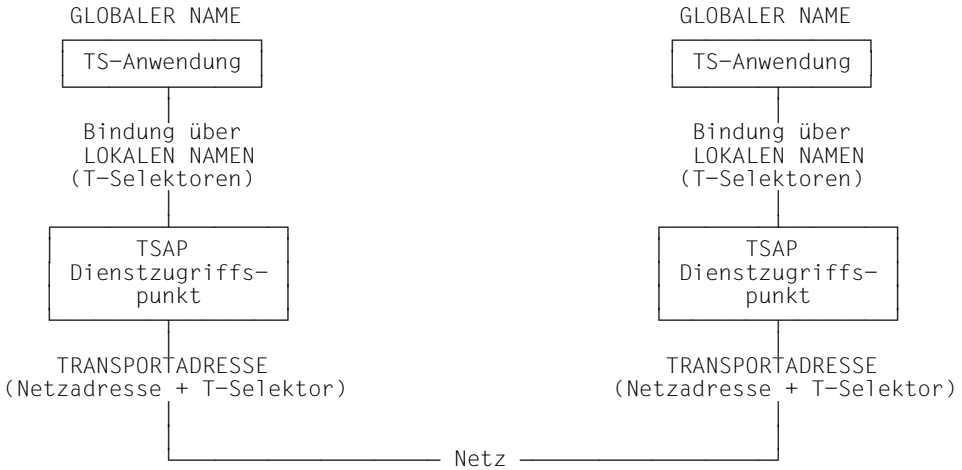
Jeder TS-Anwendung wird durch die Anmeldung bei CMX(BS2000) eindeutig ein Dienstzugriffspunkt (TSAP = Transport Service Access Point) zugeordnet. Der TSAP wird identifiziert durch den LOKALEN NAMEN, den die TS-Anwendung bei der Anmeldung bei CMX(BS2000) angibt.

Über den TSAP kann die TS-Anwendung auf die Dienste der Transportsysteme zugreifen. Auf welche Transportsysteme, d.h. Netzanschlüsse, die TS-Anwendung zugreifen kann, ist abhängig von den T-Selektoren, die der LOKALE NAME der TS-Anwendung enthält. Der LOKALE NAME enthält einen oder mehrere T-Selektoren. Ein T-Selektor kann für mehrere Netzanschlüsse gültig sein, wenn diese vom gleichen Typ sind.

Über den T-Selektor ist die TS-Anwendung aus dem Netz adressierbar, da er Bestandteil ihrer TRANSPORTADRESSE für das jeweilige Netz ist. Über die TRANSPORTADRESSE ist die TS-Anwendung im gesamten Netz eindeutig ansprechbar. Die TRANSPORTADRESSE einer TS-Anwendung setzt sich zusammen aus der Netzadresse des Endsystems, in dem die TS-Anwendung residiert, und dem T-Selektor der TS-Anwendung für diesen Netzanschluß. Die TRANSPORTADRESSE ist also wie folgt aufgebaut.

TRANSPORTADRESSE =
Netzadresse des Endsystems + (lokal eindeutiger) T-Selektor

Das folgende Bild veranschaulicht den Zusammenhang zwischen LOKALER NAME, TSAP und TRANSPORTADRESSE.



TRANSPORTADRESSE und LOKALER NAME

An der Programmschnittstelle von CMX(BS2000) stehen die Aufrufe *t_getloc(), t_getaddr()* und *t_getname()* zur Verfügung, mit denen Sie zu einem GLOBALEN NAMEN den LOKALEN NAMEN bzw. die TRANSPORTADRESSE und zu einer vorgegebenen TRANSPORTADRESSE den zugehörigen GLOBALEN NAMEN abfragen. Mit diesen Aufrufen muß die TS-Anwendung die Namen-Adreß-Umsetzung vornehmen. Der Inhalt der erzeugten Adreßstrukturen darf von der TS-Anwendung nicht ausgewertet oder verändert werden. Dadurch ist gewährleistet, daß die Anwendung von eventuellen Änderungen im Adreßaufbau nicht betroffen ist.

3.2 Struktur einer TS-Anwendung

Eine TS-Anwendung ist ein C-Programm bzw. ein System von C-Programmen, das die Funktionen von CMX(BS2000) aufruft. Was Sie bei der Erstellung eines solchen Programms beachten sollten, ist in diesem Kapitel beschrieben.

In folgendem Bild ist die Struktur eines solchen Programms angedeutet. Die angegebenen Funktionsaufrufe gehören zu der Schnittstelle ICMX.

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

        t_getloc();             /* Ermittlung LOKALER NAME */
        t_attach();             /* Anmelden bei CMX(BS2000) */

        /* 2. Kommunikationsphase */

        t_getaddr();            /* Ermitteln TRANSPORTADRESSE */
                                /* des Partners */
        t_conrq();              /* Verbindung aufbauen */
        .
        .
        t_concf();              /* Uebernahme Verb.bestaetigung */

        /* 3. Kommunikationphase */

        t_datarq();             /* Daten an den Partner senden */
        .
        .
        t_datain();             /* Daten vom Partner empfangen */
        .
        .
        t_disrq();              /* Verbindung abbauen */
        t_detach();             /* Abmelden bei CMX(BS2000) */
        .
        .
        exit();
}
```

Struktur eines TS-Anwendungsprogrammes an ICMX

Include-Datei

Jedes TS-Anwendungsprogramm muß eine Include-Anweisung für <*cmx.h*> enthalten. In <*cmx.h*> befinden sich die Definitionen der Parameter für die Funktionen der Schnittstelle ICMX.

Dieses Include ist in der Bibliothek *SYSLIB.CMX.010* enthalten.

Erlaubte Reihenfolge beim Aufruf der CMX(BS2000)-Funktionen

TS-Anwenderprogramme müssen die CMX(BS2000)-Funktionen zur Kommunikation in einer bestimmten Reihenfolge aufrufen. Der Ablauf der Kommunikation kann in drei Phasen eingeteilt werden. Die TS-Anwendung muß jede Phase erfolgreich durchlaufen, bevor sie in die nächste Phase übergehen kann.

1. Kommunikationsphase:
   Die TS-Anwendung muß sich bei CMX(BS2000) anmelden. Erst wenn CMX(BS2000) die TS-Anwendung kennt, kann diese die Leistungen von CMX(BS2000) nutzen. Die Abläufe in dieser Kommunikationsphase sind im Kapitel "An-/Abmelden bei CMX(BS2000)" beschrieben.
2. Kommunikationsphase:
   In dieser Phase stellt die TS-Anwendung die Verbindung zu ihrem Kommunikationspartner her. Bei dem Verbindungsaufbau müssen sich die beiden Partner einigen, wie der folgende Datenaustausch aussehen soll und welche Form die Daten haben. Beide Partner legen z.B. fest, ob sie auch Vorrangdaten austauschen wollen. Die Abläufe in dieser Kommunikationsphase sind im Kapitel "Verbindungen verwalten" beschrieben.
3. Kommunikationsphase:
   In der dritten Phase werden die Daten zwischen den Partnern ausgetauscht. Beide Kommunikationspartner können Daten senden und empfangen. Die Abläufe in dieser Kommunikationsphase sind im Kapitel "Daten übertragen" beschrieben.

Somit ist für ein TS-Anwendungsprogramm die Reihenfolge festgelegt, in der die CMX(BS2000)-Funktionen aufgerufen werden können. Darüberhinaus muß man beachten, daß einige Aufrufe erst erfolgen dürfen, wenn bestimmte Antworten vom jeweiligen Kommunikationspartner eingetroffen sind und von der TS-Anwendung entgegengenommen wurden (siehe Abschnitt "Ereignisverarbeitung"). Man kann sagen, daß die TS-Anwendung im Laufe der Kommunikation verschiedene Zustände einnimmt. In jeder Kommunikationsphase sind mehrere Zustände möglich. Es sind nur bestimmte Übergänge zwischen den Zuständen innerhalb einer Phase und bestimmte Übergänge zwischen Zuständen verschiedener Phasen möglich.

Eine TS-Anwendung kann nur von einem Zustand in den nächsten übergehen, indem sie bestimmte CMX(BS2000)-Funktionen aufruft oder wenn für sie bestimmte Ereignisse aus dem Netz eintreffen.
Im Abschnitt "Zustände von TS-Anwendungen und Zustandübergänge" sind dieselben in Form von Diagrammen dargestellt. Diese Diagramme sollen das Erstellen von TS-Anwendungsprogrammen erleichtern.

Verständigung über die Form der übertragenen Daten

Zwei TS-Anwendungen, die miteinander kommunizieren wollen, müssen sich auch über die Form der zu übertragenden Daten verständigen. Erforderliche Umcodierungen der Daten müssen die TS-Anwendungen selbst vornehmen, da die Übertragung durch das Transportsystem und CMX(BS2000) codetransparent ist. Dabei muß außerdem beachtet werden, welcher Zeichensatz im jeweiligen System vorliegt. In SINIX- und SINIX-ODT-Systemen ist das der ISO-7-Bit Code, in BS2000- und PDN-Systemen der EBCDIC-Code.

Parameterübergabe und Speicherbereitstellung

In TS-Anwendungen werden die Parameter zu den CMX(BS2000)-Funktionen als Werte oder Zeiger übergeben, für Optionen sind Varianten (union ...) definiert. Alle Strukturen sind in den Include-Dateien vereinbart.
Grundsätzlich müssen Sie alle Speicherbereiche, in denen Sie Werte an CMX(BS2000) übergeben, oder in die CMX(BS2000) etwas eintragen soll, in Ihrem Programm zur Verfügung stellen. Sie belegen solche Speicherbereiche entweder zur Compilierzeit (statisch) oder zur Laufzeit (dynamisch), etwa mit *malloc()* (siehe Beschreibung C-Bibliotheksfunktionen (BS2000)). In den Parameterstrukturen von CMX(BS2000) sind für Bereiche variabler Länge Längenfelder definiert. In diesen tragen Sie vor dem Aufruf von CMX(BS2000) die Länge des bereitgestellten Bereiches ein. Bei der Rückkehr können Sie daraus dann in der Regel die Länge der von CMX(BS2000) eingetragenen Daten ablesen.

3.3 Übersetzen und Binden von TS-Anwendungsprogrammen

Nachdem ein C-Programm *prog.c* einer TS-Anwendung editiert ist, muß es mit dem SIEMENS C-Compiler (ab V1.0B) übersetzt (compiliert) und die CMX(BS2000)-Funktionen aus der CMX(BS2000)-Bibliothek *SYSLIB.CMX.010* ins Programm eingebunden werden. Außerdem muß dazu auch das C-RTS eingebunden werden. Aus der Bibliothek *SYSLIB.CMX.010* muß der Modul *YDCMXLNK* zum Programm gebunden werden. Dieser Modul nimmt den Anschluß an das Subsystem CMX-TU vor, welches die eigentliche CMX-Bibliothek realisiert.

Vorteile hiervon sind:

– Die gebundenen Programme sind wesentlich kleiner.

– Nach (evtl. wartungsbedingtem) Austausch einer Bibliothek müssen die Anwendungsprogramme nicht neu gebunden werden.

Weitere Informationen über Subsysteme entnehmen Sie bitte den Systemmanualen oder der Freigabemitteilung.

Hinweis

In dem Modul *YDCMXLNK* legt CMX taskspezifische Daten ab. Er darf daher nicht schreibgeschützt werden und nicht zu SHARED CODE-Moduln gebunden werden.

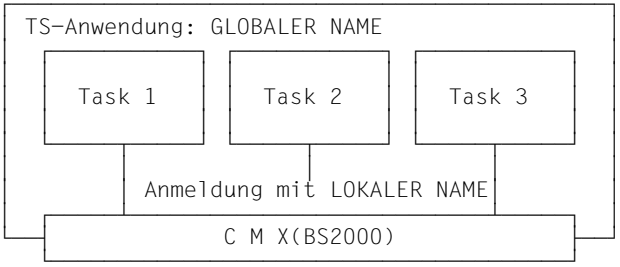
3.4 TS-Anwendungen, Tasks, Verbindungen

Die beiden folgenden Abschnitte beschreiben die Zusammenhänge TS-Anwendung - Tasks und Tasks - Verbindungen.

3.4.1 TS-Anwendungen und Tasks

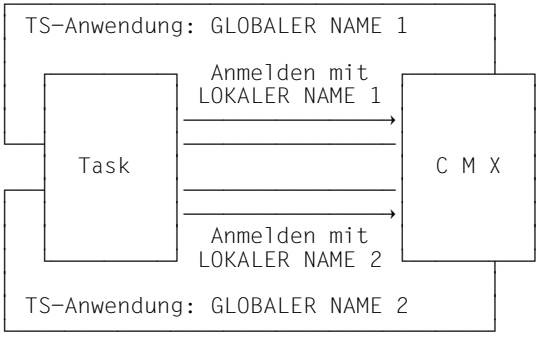
Im einfachsten Fall ist eine TS-Anwendung in einer einzigen Task realisiert. Zur Strukturierung einer TS-Anwendung gibt es aber weitere Möglichkeiten.

Eine TS-Anwendung kann in mehreren Tasks arbeiten. Die Tasks müssen nicht unter der gleichen USER-ID (Kennung) gestartet werden, können also unter verschiedenen Kennungen laufen. Jede einzelne Task einer TS-Anwendung, muß sich bei CMX(BS2000) anmelden. Tasks gehören zur selben TS-Anwendung, wenn sie sich mit demselben LOKALEN NAMEN bei CMX(BS2000) angemeldet haben. Die erste Task, die sich anmeldet, erzeugt die TS-Anwendung.



Eine TS-Anwendung - mehrere Tasks

Andererseits kann eine Task mehrere TS-Anwendungen steuern. Dazu meldet man die Task mit verschiedenen LOKALEN NAMEN bei CMX(BS2000) an.



Eine Task - mehrere TS-Anwendungen

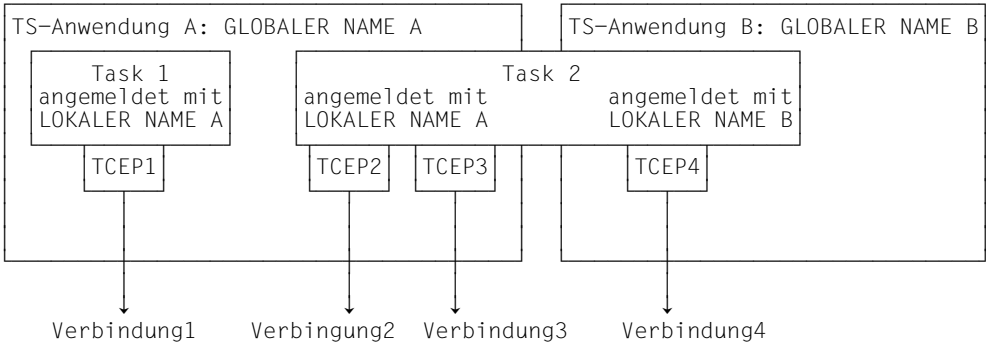
Die Task unterscheidet die verschiedenen TS-Anwendungen, die sie steuert, durch die verschiedenen LOKALEN NAMEN oder eine frei wählbare Benutzerreferenz.

Im CMX(BS2000)-Konzept sind asynchrone Routinen ("Contingencies") nicht vorgesehen (analog zu SINIX). Falls Contingencies doch verwendet werden, muß man darauf achten, daß zu einem Zeitpunkt nur ein CMX-Aufruf in der Bibliothek bearbeitet wird. Ausnahme: *t_wake*.

3.4.2 Verbindungen und Tasks

Die Tasks einer TS-Anwendung können unabhängig voneinander Verbindungen zu anderen TS-Anwendungen aufbauen. Dabei kann eine einzelne Task gleichzeitig mehrere Verbindungen halten. Die Verbindungen können auch zu verschiedenen TS-Anwendungen gehören, wenn die Task in mehreren TS-Anwendungen angemeldet ist. Beim Verbindungsaufbau wird ein Transportendpunkt (TCEP - Transport Connection Endpoint) für jede Verbindung eingerichtet. Eine Task kann also mehrere TCEP bedienen. Ein TCEP kann jedoch nicht gleichzeitig mehreren Tasks zugeordnet sein. Jede Verbindung ist zu jedem Zeitpunkt **genau** einer Task zugeordnet.

Jede Verbindung erhält von CMX(BS2000) eine Identifikation, die Transportreferenz. Damit allein kann die Task eine Verbindung gezielt ansprechen.



Verbindungen und Tasks

Eine Task kann aber eine Verbindung zu einer anderen Task, die sich bereits in derselben TS-Anwendung angemeldet hat, umlenken. Die Verbindung ist dann der Task, die sie abgegeben hat, nicht mehr bekannt. Damit kann man Verbindungen zu verschiedenen Partnern in verschiedenen Tasks behandeln. Ein zentraler Verteilprozeß kann z.B. alle Verbindungen entgegennehmen und dann an geeignete nachgeordnete Tasks umlenken. Im Bild oben kann z.B. die Task 2 die Verbindung 2 oder 3 an die Task 1 umlenken.

4 Ereignisverarbeitung und Fehlerbehandlung

Dieses Kapitel beschreibt die Ereignisverarbeitung und Fehlerbehandlung bei TS-Anwendungen mit CMX(BS2000).

4.1 Ereignisverarbeitung

Die Vorgänge bei der Kommunikation von TS-Anwendungen sind asynchron, d.h. während der Kommunikation können unabhängig vom Verhalten der TS-Anwendung die verschiedensten Ereignisse eintreten. Ereignisse sind Anforderungen und Antworten anderer TS-Anwendungen im Netz, die CMX(BS2000) empfangen hat, bzw. Mitteilungen der beteiligten Transportsysteme.

Beispiele für solche Ereignisse sind:

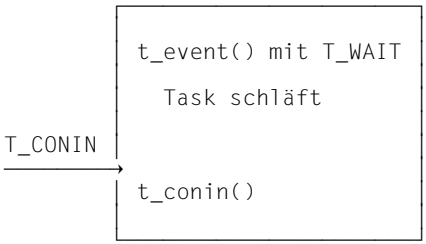
– der Verbindungswunsch eines Kommunikationspartners (der "rufenden Anwendung")
– die Ankunft von Daten auf einer bestehenden Verbindung
– Flußregelungsereignisse (Sendesperre gesetzt bzw. aufgehoben)
– der Verbindungsabbau durch den Kommunikationspartner oder CMX(BS2000)

CMX(BS2000) stellt diese Ereignisse der TS-Anwendung zu, wenn die TS-Anwendung die Funktion *t_event()* aufruft. Bei jedem Aufruf *t_event()* übergibt CMX(BS2000) genau ein Ereignis gegebenenfalls zusammen mit der Identifikation der betroffenen Verbindung (Transportreferenz). Die TS-Anwendung muß das entgegengenommene Ereignis dann sofort entsprechend verarbeiten, z.B. die entsprechende "abholende Funktion" aufrufen.

Die Funktionen von CMX(BS2000) sind so ausgelegt, daß die TS-Anwendung nach Absetzen eines Aufrufs auf die eventuelle Antwort aus dem Netz warten kann, aber nicht muß. Sie hat zwei Möglichkeiten Ereignisse zu verarbeiten.

1. Synchrone Verarbeitung

Die TS-Anwendung ruft *t_event()* mit Parameter *cmode* = T_WAIT auf. Solange kein Ereignis ansteht, schläft die Task und verbraucht keine Rechenzeit. Bei einem Ereignis (im Bild T_CONIN) weckt CMX(BS2000) die Task auf, und *t_event()* bringt als Ergebnis den Code des Ereignisses sowie gegebenenfalls die Transportreferenz der betroffenen Verbindung.



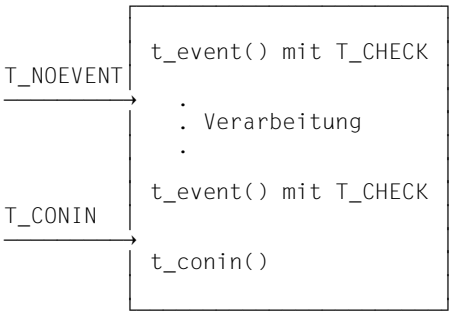
Synchrone Verarbeitung

Auch wenn die Task in *t_event()* schläft, kann man sie mit *t_wake* aufwecken. CMX(BS2000) setzt ihn dann mit T_NOEVENT fort. Beim Aufruf von *t_event()* kann man auch die Wartezeit beschränken. Man gibt einfach an, wie lange die Task auf ein Ereignis warten soll. Trifft in dieser Zeit kein Ereignis ein, setzt CMX(BS2000) die Task mit T_NOEVENT fort.

2. Asynchrone Verarbeitung

Sie rufen *t_event()* mit Parameter cmode = T_CHECK auf. Falls kein Ereignis ansteht, kehrt der Aufruf sofort mit T_NOEVENT zurück. Sie können mit beliebiger Verarbeitung fortfahren und später *t_event()* erneut aufrufen, um ein eventuelles Ereignis abzufragen.

Es ist aber nicht sinnvoll, nur *t_event()* in einer dauernden Schleife laufen zu lassen, besser sollten Sie dann die synchrone Ereignisverarbeitung (cmode = T_WAIT) verwenden.



Asynchrone Verarbeitung

Abhängig davon, welches Ereignis gemeldet wurde, erwartet CMX(BS2000) eine bestimmte Reaktion. Da der Programmablauf davon abhängt, welche Ereignisse auftreten, kann man die Programmlogik weitgehend in einer switch-Konstruktion verpacken, deren case's die verschiedenen Ereignisse sind (wie in den Beispielprogrammen).

4.2 Fehlerbehandlung

Ein fehlerhaft abgelaufener Funktionsaufruf (t_ ...) kehrt immer mit einer globalen Fehleranzeige (T_ERROR) zurück. Einen detaillierten Wert erhält man durch den Aufruf einer Fehlerabfragefunktion (t_error()).

Die von *t_error()* gelieferten Werte sind hexadezimal codiert und dienen der Diagnose.

Im Anhang finden Sie den Aufbau der CMX(BS2000)-Fehlermeldungen, eine Tabelle mit den CMX-Fehlerwerten sowie Tabellen mit der Zuordnung der CMX-Fehlerwerte zu den BCAM-Returncodes und die Bedeutung der einzelnen Returncodes.

5 An-/Abmelden bei CMX(BS2000)

Eine TS-Anwendung entsteht, sobald sich eine Task mit dem LOKALEN NAMEN der TS-Anwendung bei CMX(BS2000) anmeldet. Jede weitere Task, die in dieser TS-Anwendung arbeiten will, muß sich ebenfalls für diese TS-Anwendung, d.h. mit ihrem LOKALEN NAMEN, bei CMX(BS2000) anmelden.

Bevor sich eine Task beendet, muß sie alle ihre TS-Anwendungen bei CMX(BS2000) abmelden. Hat sich die letzte Task einer TS-Anwendung bei CMX(BS2000) abgemeldet, so existiert diese TS-Anwendung für CMX(BS2000) nicht mehr.

5.1 Anmelden bei CMX(BS2000)

Die Anmeldung einer Task bei CMX(BS2000) über die Programmschnittstelle ICMX erfolgt mit dem Aufruf *t_attach()*.

Bei der Anmeldung muß die Task den LOKALEN NAMEN der TS-Anwendung übergeben, für die sie sich bei CMX(BS2000) anmelden will. Den LOKALEN NAMEN muß die Task vor der Anmeldung, d.h. vor dem Aufruf von *t_attach()*, mit Hilfe der ICMX-Funktion *t_getloc()* ermitteln. Sie übergibt *t_getloc()* als Parameter den GLOBALEN NAMEN der TS-Anwendung, für die sie sich anmelden will. *t_getloc()* liefert den Zeiger auf eine Struktur zurück, in der der LOKALE NAME steht. Dieser Zeiger wird beim *t_attach()* als Parameter übergeben. Der Aufruf *t_getloc()* muß also vor dem *t_attach()* erfolgen.

Bei der Anmeldung der ersten Task einer TS-Anwendung wird ein Dienstzugriffspunkt (Transport Service Access Point = TSAP) für diese TS-Anwendung eingerichtet. An dem TSAP steht der Transportservice zur Verfügung. Dem TSAP wird der LOKALE NAME der TS-Anwendung zugeordnet.

Jede Task einer TS-Anwendung kann:

– für die TS-Anwendung aktiv Verbindungen aufbauen. Die TS-Anwendung kann dann in der folgenden Verbindungsaufbauphase die Funktion der "rufenden TS-Anwendung" übernehmen.

– für die TS-Anwendung passiv auf Verbindungsanforderungen von anderen TS-Anwendungen im Netz warten. Die TS-Anwendung kann dann im Verlauf der Kommunikation die Funktion der "gerufenen TS-Anwendung" übernehmen.

– Verbindungen annehmen, die eine andere Task der gleichen TS-Anwendung an sie übergeben will (Verbindungsumlenkung annehmen). Eine Task der gleichen TS-Anwendung ist eine Task, die sich mit demselben LOKALEN NAMEN bei CMX(BS2000) angemeldet hat.

Dieselbe Task kann sich auch für mehrere verschiedene TS-Anwendungen anmelden. Dazu ruft sie *t_getloc()* und *t_attach()* für jede dieser TS-Anwendungen auf.

CMX(BS2000) nimmt für eine TS-Anwendung Verbindungsanforderungen von entfernten TS-Anwendungen entgegen, sobald sich eine Task der TS-Anwendung bei CMX(BS2000) angemeldet hat. Ankommende Verbindungswünsche gibt CMX(BS2000) zunächst an die Task weiter, die sich als erste in dieser TS-Anwendung angemeldet hat.

Erst nach erfolgreicher Anmeldung kann die Task andere CMX(BS2000)-Funktionen aufrufen, d.h. andere *t_...()*-Aufrufe absetzen.

5.2 Abmelden bei CMX(BS2000)

Bevor sich eine Task beendet, ruft sie *t_detach()* auf. *t_detach()* meldet die Task bei CMX(BS2000) für die TS-Anwendung ab. Zuvor müssen aber alle TS-Verbindungen, die diese Task hält, abgebaut werden (siehe Kapitel "Verbindungen verwalten"). Tut die Task das nicht, baut CMX(BS2000) implizit selbst alle TS-Verbindungen ab. Dies ist aber nur für Ausnahmesituationen vorgesehen, z.B. wenn sich eine Task unerwartet frühzeitig beendet.

Sobald sich die letzte Task aus einer TS-Anwendung abgemeldet hat, existiert diese TS-Anwendung für CMX(BS2000) nicht mehr. Verbindungsanforderungen von fernen TS-Anwendungen werden für diese TS-Anwendung nicht mehr angenommen.

5.3 Beispiel zur An- und Abmeldung einer Task

Beispiel zu An- und Abmelden bei ICMX

Das folgende Programmfragment zeigt den Programmablauf zum Anmelden und Abmelden einer Task an ICMX.
Eine Task meldet sich für die TS-Anwendung "TestanwendungAKT" bei CMX(BS2000) an und wieder ab. In der Optionsstruktur *t_opta2* gibt sie an, daß sie in dieser TS-Anwendung nur aktiv Verbindungen aufbauen will (T_ACTIVE) und gleichzeitig maximal 1 Verbindung halten will. Diese Angabe wird von der CMX(BS2000) Version 1.0 jedoch ignoriert, d.h., es können mehrere Verbindungen sowohl aktiv als auch passiv aufgebaut werden.

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

#define MYNAME  "TestanwendungAKT"
char *myname = { MYNAME } ;
struct t_myname t_myname, *p_myname;
   .
   .
/* Aktive Anwendung bei CMX(BS2000) anmelden */

if ((p_myname = t_getloc(myname, NULL)) != NULL)
        t_myname = *p_myname;
else {
       fprintf(stderr, ">>> FEHLER 0x%x bei t_getloc\n", t_error());
       exit(ERROR);
}
if (t_attach(&t_myname, &t_opta2) == T_ERROR) {
       fprintf(stderr, ">>> FEHLER 0x%x bei t_attach\n", t_error());
       exit(ERROR);
}
fprintf(stderr, "Anwendung '%s' angemeldet.\n", myname);
  .
  .

/* TS-Anwendung bei CMX(BS2000) abmelden */
if (t_detach(&t_myname) == T_ERROR)
        fprintf(stderr, ">>> FEHLER 0x%x bei t_detach\n", t_error());
fprintf(stderr, "Anwendung '%s' abgemeldet.\n", myname);
   .
   .
```

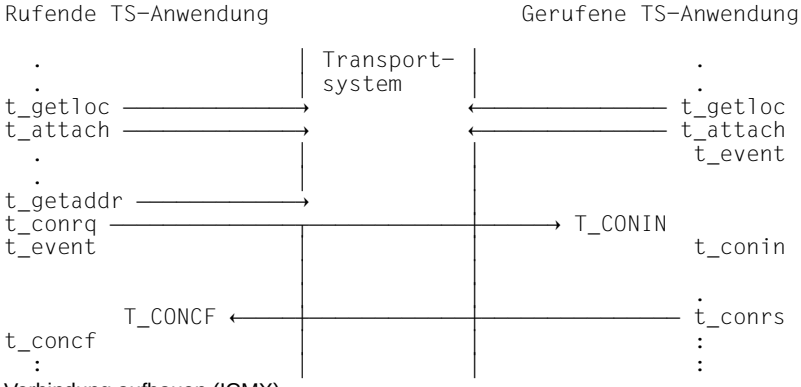
6 Verbindungen verwalten

Verbindungsaufbau und -abbau laufen zwischen zwei TS-Anwendungen ab. Die eine ist die rufende TS-Anwendung, sie initiiert den Verbindungsaufbau. Die andere ist die gerufene TS-Anwendung, mit der die rufende TS-Anwendung eine Verbindung eingehen will. Die folgenden Abschnitte verdeutlichen die Zusammenhänge und Abläufe.

Daß in den Diagrammen CMX(BS2000) nur einmal dargestellt ist, ist nur eine Vereinfachung der Darstellung. Tatsächlich benutzt jeder Partner "sein" CMX(BS2000) in seinem Rechner und dazwischen liegen das Netzwerk und die Transportsysteme.

6.1 Verbindung aufbauen

Zunächst werden die Abläufe beim Verbindungsaufbau an ICMX betrachtet. Das folgende Bild zeigt den zeitlichen Ablauf der ICMX-Aufrufe in den Programmen der rufenden und der gerufenen TS-Anwendung.



Verbindung aufbauen (ICMX)

Ablauf des Verbindungsaufbaus in der rufenden TS-Anwendung

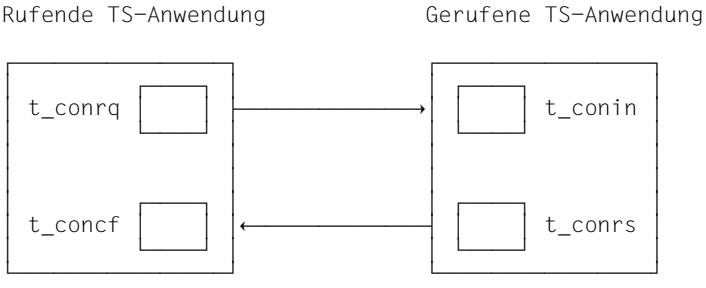
Die rufende TS-Anwendung holt sich zuerst ihren LOKALEN NAMEN und meldet sich dann bei CMX(BS2000) an. Danach ermittelt sie die TRANSPORTADRESSE der gerufenen TS-Anwendung mit *t_getaddr()* und fordert mit *t_conrq()* einen Verbindungsaufbau an. Dann wartet sie mit *t_event()* auf die Bestätigung der gerufenen TS-Anwendung, d.h. auf das TS-Ereignis T_CONCF. Wenn *t_event()* das TS-Ereignis gemeldet hat, stellt die rufende TS-Anwendung die Verbindung mit dem Aufruf *t_concf()* her.

Ablauf des Verbindungsaufbaus in der gerufenen TS-Anwendung

Die gerufene TS-Anwendung wartet nach der Anmeldung zunächst mit *t_event()* auf ein TS-Ereignis. Das TS-Ereignis T_CONIN zeigt den Verbindungsaufbauwunsch der rufenden TS-Anwendung an. Mit dem Aufruf *t_conin()* nimmt die gerufene TS-Anwendung den Verbindungsaufbauwunsch an und beantwortet ihn mit *t_conrs()*.

Benutzerdaten beim Verbindungsaufbau austauschen

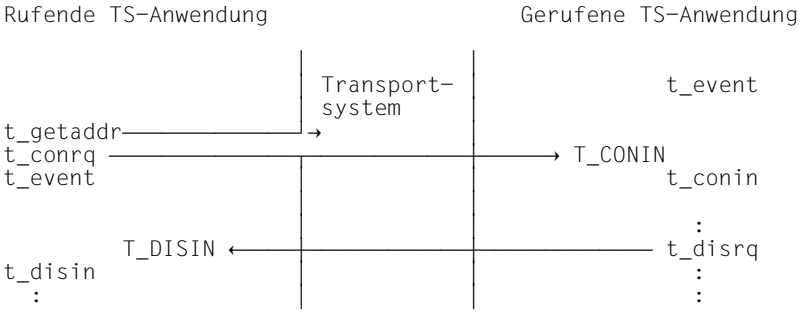
Die Aufrufe *t_conin()* (Entgegennehmen der Verbindungsaufbauanzeige) und *t_concf()* (Herstellen der Verbindung) sind nötig, da beide TS-Anwendungen schon beim Verbindungsaufbau Benutzerdaten austauschen können (siehe Abschnitt "Optionen des Systems und des Benutzers").
Bei *t_conrq()* kann die rufende TS-Anwendung Benutzerdaten mitgeben. Das ist eine kleine Datenmenge, die die gerufene TS-Anwendung bei *t_conin()* erhält. Wenn die gerufene TS-Anwendung dann den Verbindungswunsch mit *t_conrs()* beantwortet, kann sie wiederum Informationen mitgeben. Diese erhält die rufende TS-Anwendung bei *t_concf()*.



Benutzerdatenaustausch beim Verbindungsaufbau

Verbindungswunsch ablehnen

Die gerufene TS-Anwendung kann den Verbindungswunsch auch ablehnen. Der Ablauf ist derselbe.
Das Ereignis T_CONIN muß zunächst mit *t_conin()* angenommen werden, statt des Aufrufes *t_conrs()* ist aber *t_disrq()* zu verwenden (siehe auch Verbindungen abbauen).



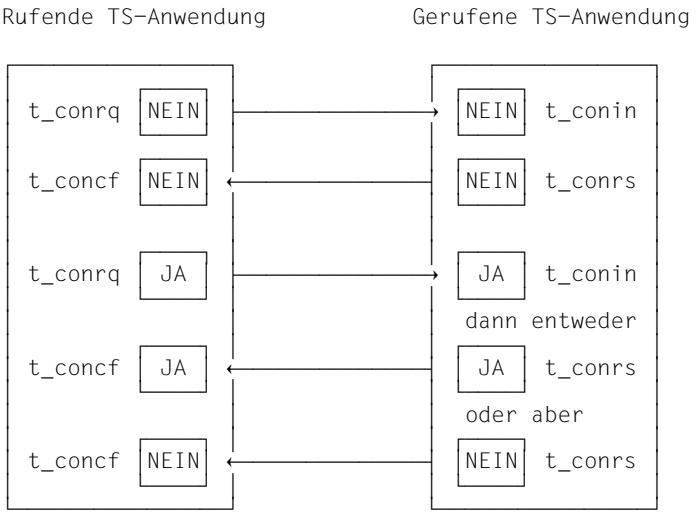
Verbindungswunsch ablehnen

Vorrangdaten aushandeln

Können über die aufzubauende Transportverbindung Vorrangdaten übertragen werden, dann können die TS-Anwendungen beim Verbindungsaufbau deren Verwendungen aushandeln. Das geht wie folgt:

Die rufende TS-Anwendung macht beim Verbindungswunsch mit *t_conrq()* einen Vorschlag, den die gerufene TS-Anwendung nur "herunterhandeln" kann. Das bedeutet: schlägt die rufende TS-Anwendung vor, keine Vorrangdaten zu verwenden, so ist dies für die Verbindung verbindlich. Schlägt sie dagegen vor, Vorrangdaten auszutauschen, so kann die gerufene TS-Anwendung bei der Verbindungsbeantwortung mit *t_conrs()* zustimmen oder ablehnen. In beiden Fällen ist die Antwort verbindlich.

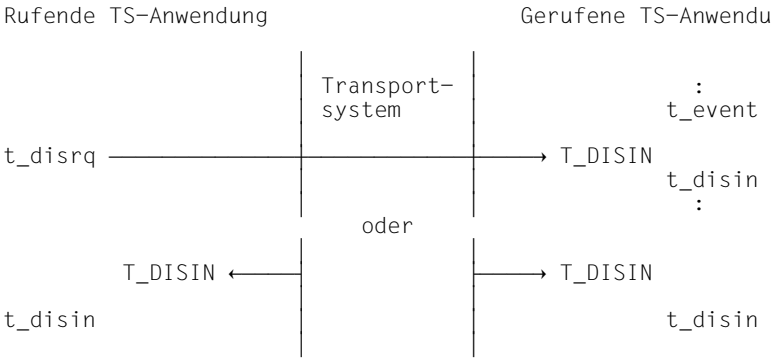
Wenn eine der beiden TS-Anwendungen mit dem Ergebnis der Vorrangdatenverhandlung nicht einverstanden ist, kann sie die Verbindung abbauen.



Vorrangdatenverhandlung beim Verbindungsaufbau

6.2 Verbindung abbauen

Jede der beiden kommunizierenden TS-Anwendungen kann *t_disrq()* aufrufen, um die Verbindung abzubauen. Die Partner-TS-Anwendung erhält das Ereignis T_DISIN. Mit dem Aufruf *t_disin()* nimmt diese den Verbindungsabbau entgegen. Dabei erfährt sie den Grund für den Verbindungsabbau.



Verbindung abbauen

Wenn die Transportverbindung die entsprechende Option bietet, kann die TS-Anwendung, die die Verbindung abbaut, bei *t_disrq()* Benutzerdaten mitschicken. Die Partner-TS-Anwendung erhält sie bei *t_disin()*.

Auch das Transportsystem kann die Verbindung abbauen. Beide TS-Anwendungen erhalten dann das Ereignis T_DISIN, das sie mit *t_disin()* abholen müssen. Aus dem Verbindungsabbaugrund kann jede von ihnen ermitteln, ob die andere TS-Anwendung oder das Transportsystem die Verbindung abgebaut hat.

6.3 Beispiele zum Verbindungsaufbau und -abbau mit ICMX

Die beiden folgenden Programmfragmente zeigen, wie man eine Verbindung aufbaut. Beispiel 1 zeigt den Programmaufbau für die rufende TS-Anwendung, Beispiel 2 zeigt den Programmaufbau für die gerufene TS-Anwendung.

Beispiel 1

Die TS-Anwendung baut aktiv eine Verbindung zu der TS-Anwendung "TestanwendungPAS" auf und wieder ab.

```
#include        <stdio.h>
#include        <cmx.h>
   .
   .
#define ERROR   1
   .
   .
int     tref;                   /* Transportreferenz */
int     reason;                 /* Grund für Verbindungsabbau */

 /* Strukturen für Adressierung */

#define PNAME   "TestanwendungPAS"
char *pname = { PNAME } ;
struct t_partaddr t_partaddr, *p_partaddr;
   .
   .
/* Verbindung aufbauen zum passiven Partner */

if ((p_partaddr = t_getaddr(pname, NULL)) != NULL)
        t_partaddr = *p_partaddr;
else {
      fprintf(stderr, ">>> FEHLER 0x%x bei 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, ">>> FEHLER 0x%x bei t_conrq, tref 0x%x\n",
              t_error(), tref);
      exit(ERROR);
}
```

```
/* Ereignisgesteuerte Verarbeitung:
 * t_event() synchron (T_WAIT) wartend
 */
for (;;) {
        switch (event = t_event(&tref, T_WAIT, NULL)) {
        case T_CONCF:
           /*
            * Verbindungsaufbau gelungen?
            */
           if (t_concf(&tref, NULL) == T_ERROR) {
                  fprintf(stderr, ">>> FEHLER 0x%x bei t_concf tref 0x%x\n",
                        t_error(), tref);
                  exit(ERROR);
           }
           fprintf(stderr, "Verbindung zu '%s' aufgebaut.\n",
                   pname);
           .
           .
        case T_DISIN:

            /* Verbindungsabbau durch Partner oder System */

            if (t_disin(&tref, &reason, NULL) == T_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei t_disin tref 0x%x\n",
                        t_error(), tref);
                exit(ERROR);
            }
                 fprintf(stderr, "Verbindungsabbau erhalten, tref 0x%x, reason
%d\n",
                   tref, reason);
            .
            .
        }
}
 /* Verbindungsabbau */

 if (t_disrq(&tref, NULL) == T_ERROR){
            fprintf(stderr, ">>> FEHLER 0x%x bei t_disrq tref 0x%x\n",
                    t_error(), tref);
            exit(ERROR);
 }
 fprintf(stderr, "Verbindung tref 0x%x aktiv abgebaut.\n", tref);
    .
    .
```

Beispiel 2

Die TS-Anwendung wartet passiv auf eine eintreffende Verbindungsanforderung, nimmt die Verbindung an und baut sie wieder ab.

```
#include        <stdio.h>
#include        <cmx.h>
   .
   .
#define ERROR   1
   .
   .
int     tref;                   /* Transportreferenz */
int     reason;                 /* Grund für Verbindungsabbau */
/*
 * Strukturen für Adressierung
 */
struct t_myname t_myname, *p_myname;
struct t_partaddr t_partaddr;
   .
   .
   .
/* Ereignisgesteuerte Verarbeitung:
 * t_event() synchron (T_WAIT) wartend
 */
for (;;) {
        switch (event = t_event(&tref, T_WAIT, NULL)) {

        case T_CONIN:

            /* Verbindungsaufbauwunsch akzeptieren */

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

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

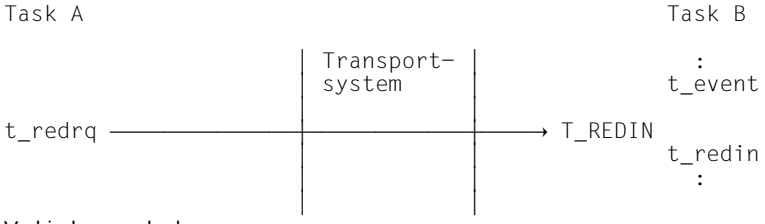
```
        case T_DISIN:
            /*
             * Verbindungsabbau durch Partner oder System
             */
            if (t_disin(&tref, &reason, NULL) == T_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei t_disin tref 0x%x\n",
                        t_error(), tref);
                exit(ERROR);
             }
                 fprintf(stderr, "Verbindungsabbau erhalten, tref 0x%x, reason
%d\n",
                    tref, reason);
                .
                .
        }
}
/*
 * Verbindungsabbau
 */
if (t_disrq(&tref, NULL) == T_ERROR){
    fprintf(stderr, ">>> FEHLER 0x%x bei t_disrq tref 0x%x\n",
            t_error(), tref);
    exit (ERROR);
}
fprintf(stderr, "Verbindung tref 0x%x aktiv abgebaut.\n", tref);
    .
    .
```

6.4 Verbindungen umlenken

Ankommende Verbindungen für eine lokale TS-Anwendung erhält zunächst die Task, die sich als erste für diese TS-Anwendung angemeldet hat. Um nun z.B. bestimmte Verbindungen bestimmten Tasks zuordnen zu können, kann man eine Verbindung an eine andere Task weitergeben. Man kann natürlich auch aktiv aufgebaute Verbindungen weitergeben. Beide Tasks müssen zur selben TS-Anwendung gehören, das heißt, sich mit demselben LOKALEN NAMEN angemeldet haben.

Ablauf beim Umlenken einer Verbindung

Task A gibt beim Aufruf *t_redrq()* die TSN von Task B an. Task B erhält das Ereignis T_REDIN zugestellt und muß mit dem Aufruf *t_redin()* die Verbindung zunächst annehmen. Bei diesem Aufruf erhält Task B die TSN von Task A mitgeteilt. Will Task B die Verbindung nicht haben, kann sie sie abbauen oder weiter umlenken, z.B zurück an Task A.



Verbindung umlenken

Man kann auch bei *t_redrq()* Benutzerdaten mitgeben, die Task B beim Aufruf *t_redin()* erhält.

6.5 Beispiel zur Umlenkung einer Verbindung

Die folgenden Programmfragmente zeigen, wie man eine Verbindung umlenkt und eine umgelenkte Verbindung entgegenimmt.

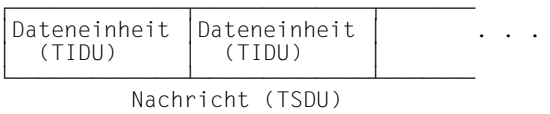
```
#include        <stdio.h>
#include        <cmx.h>
  .
  .
#define ERROR   1
  .
  .
int     tref;     /* Transportreferenz */
int     cpid;     /* ID der Task, die Verbindung erhalten soll */
int     rpid;     /* ID der Task, die Verbindung abgeben will */
  .
  .
/* Verbindung aktiv umlenken */

if (t_redrq(&tref, &cpid, NULL) == T_ERROR) {
    fprintf(stderr, ">>> FEHLER 0x%x bei t_redrq tref 0x%x\n",
            t_error(), tref);
    exit(ERROR);
}

/* Verbindungsumlenkung entgegennehmen */

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

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

         }
}
```

7 Daten übertragen

Sobald eine Verbindung aufgebaut ist, können die beiden TS-Anwendungen Daten austauschen. Beide TS-Anwendungen können mit dem Datenaustausch beginnen, unabhängig davon, ob es die rufende oder gerufene TS-Anwendung ist.

Die Gesamtmenge von Daten, die aus der Sicht der TS-Anwendungen eine logische Einheit bilden, wird als Nachricht bezeichnet oder als TSDU (Transport Service Data Unit). Die Länge einer TSDU ist beliebig.

CMX(BS2000) kann jedoch immer nur eine begrenzte Datenmenge auf einmal annehmen. Diese bezeichnet man als Dateneinheit oder TIDU (Transport Interface Data Unit). Wie lang eine TIDU höchstens sein darf, hängt von der Transportverbindung ab. Die Länge muß man mit dem Aufruf *t_info()* für jede Verbindung abfragen.



TIDU und TSDU

Die logische Verknüpfung der TIDUs zu einer TSDU wird durch einen Parameter gesteuert, der bei jeder TIDU einer Nachricht angibt, ob ihr eine weitere TIDU folgt oder ob sie die letzte der TSDU ist.

Wenn die Transportverbindung diese Option bietet und beide TS-Anwendungen dies beim Verbindungsaufbau vereinbart haben, können sie auch Vorrangdaten austauschen. Vorrangdaten sind kleine Datenmengen, die mit "Vorrang" vor den Normaldaten transportiert werden. D.h. Vorrangdaten treffen nie später ein als danach gesendete Normaldaten. Vorrangdaten können immer nur auf einmal übertragen werden. Die Einheit der Vorrangdaten heißt ETSDU (Expedited Transport Service Data Unit).

7.1 Senden und Empfangen von Normaldaten

Normaldaten sendet man mit einem der Aufrufe *t_datarq()* oder *t_vdatarq()*.

Jeder solche Aufruf sendet eine TIDU. *t_datarq()* wird aufgerufen, wenn die zu sendende TIDU in einem zusammenhängenden Speicherbereich steht. *t_vdatarq()* wird aufgerufen, wenn die zu sendende TIDU in mehreren Teilbereichen des Speichers bereitgestellt wird.

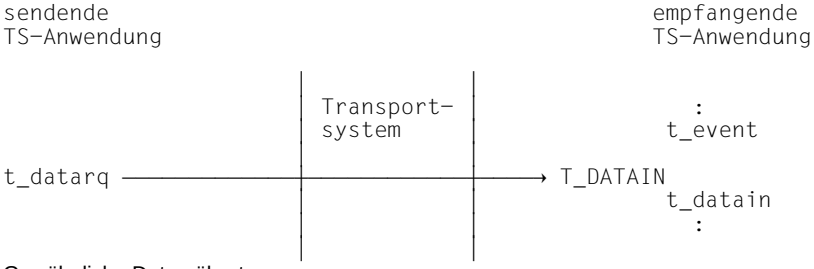
Im einfachsten Fall läuft die Datenübertragung so ab:

– Die sendende TS-Anwendung übergibt CMX(BS2000) mit jedem Aufruf eine TIDU.

– Die empfangende TS-Anwendung erhält das Ereignis T_DATAIN angezeigt. Das ist die Mitteilung, daß Daten angekommen sind.

– Die empfangende TS-Anwendung muß die Daten mit dem Aufruf *t_datain()* oder mit dem Aufruf *t_vdatain()* annehmen.

t_datain() und *t_vdatain()* unterscheiden sich dadurch, daß die Daten bei *t_datain()* in einen zusammenhängenden Speicherbereich und bei *t_vdatain()* in mehrere, nicht zusammenhängende Speicherbereiche übernommen werden.



Gewöhnliche Daten übertragen

Wenn die TSDU länger ist als eine maximale TIDU ...

muß sie in TIDUs zerlegt werden. Das geht so:

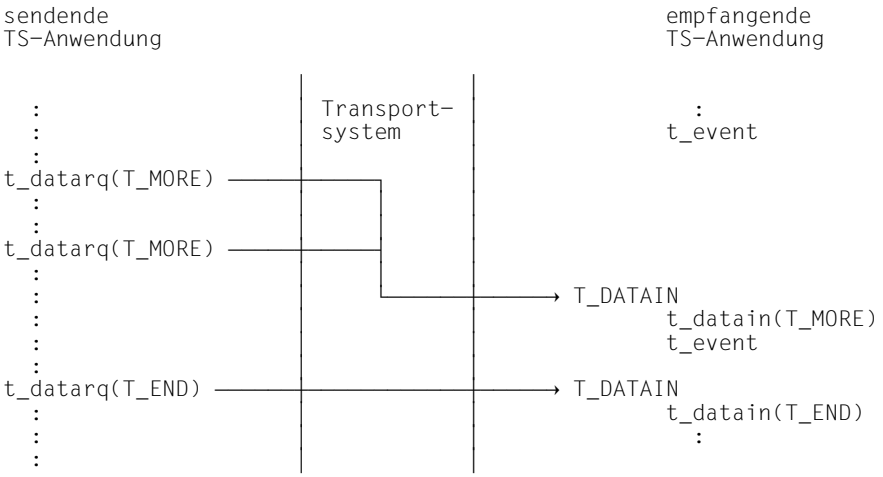
– Die sendende TS-Anwendung bestimmt als Sender, wann die TSDU zu Ende ist. Bei jedem Absenden einer TIDU mit *t_datarq()* oder *t_vdatarq()* gibt sie im Parameter *chain* an, ob eine weitere TIDU in dieser TSDU folgt (*chain* = T_MORE) oder die gerade zu sendende TIDU die letzte ist (*chain* = T_END).

– Die empfangende TS-Anwendung bekommt in gleicher Weise bei jedem Aufruf *t_datain()* oder *t_vdatain()* in *chain* mitgeteilt, ob noch eine weitere TIDU in dieser TSDU folgt.

Jede TIDU kündigt CMX(BS2000) mit einem Ereignis T_DATAIN an, aber:

TIDU ist nicht gleich TIDU!

Die Länge einer TIDU kann bei beiden TS-Anwendungen unterschiedlich sein. Deshalb kann es sein, daß die empfangende TS-Anwendung weniger oft *t_datain()* bzw. *t_vdatain()* aufrufen muß als die sendende TS-Anwendung *t_datarq()* bzw. *t_vdatarq()* (oder umgekehrt). Denn die empfangende TS-Anwendung liest TIDUs in "ihrer" Länge. Das sieht dann wie folgt aus:



TSDU in mehreren TIDUs

Der Ergebniswert von t_datain() und t_vdatain()

Bei *t_datain()* und *t_vdatain()* muß eine Länge angegeben werden, in der die angekommenen Daten gelesen werden sollen. Wenn die angegebene Länge kleiner ist als die angezeigte TIDU beim Empfänger, gibt der Ergebniswert von *t_datain()* und *t_vdatain()* die Restlänge der Daten in der anstehenden TIDU an.

Ist eine TIDU noch nicht vollständig gelesen, muß erneut *t_datain()* bzw. *t_vdatain()* aufgerufen werden, und zwar so oft, bis die TIDU vollständig gelesen ist. Währenddessen darf auch nicht *t_event()* aufgerufen, die Verbindung umgelenkt oder der Datenfluß geregelt werden.

Zu beachten ist, daß CMX(BS2000) nicht garantiert, daß bei der empfangenden TS-Anwendung alle TIDUs einer Nachricht maximal gefüllt sind, selbst dann nicht, wenn die TIDU bei sendender und empfangender TS-Anwendung gleich ist und die sendende TS-Anwendung nur maximal gefüllte TIDUs sendet.

7.2 Beispiel zur Übertragung von Normaldaten

Die folgenden Programmfragmente zeigen den Programmablauf bei der Übertragung von Normaldaten über ICMX.

Die TS-Anwendung empfängt und sendet Daten. Die Länge der Daten ist hier auf eine TIDU beschränkt.

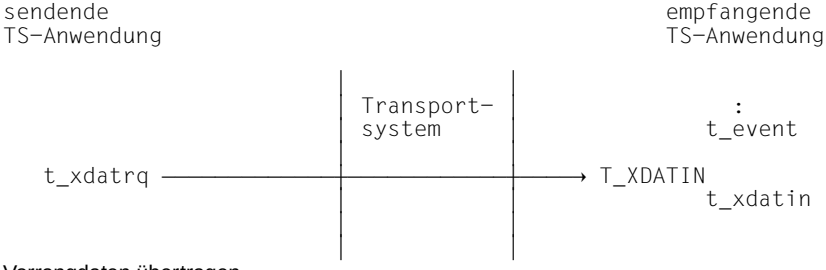
```
#include        <stdio.h>
#include        <cmx.h>
.
.
#define ERROR   1
.
.
/* Sende- und Empfangspuffer */

char    e_bufpt[8000];          /* Empfangspuffer */
int     e_bufl;                 /* Übertragungslänge */
char    s_bufpt[8000];          /* Sendepuffer */
int     s_bufl;                 /* Übertragungslänge */

int     chain;                  /* TSDU-Indikator für t_datarq(), t_datain() */
int     tref;                   /* Transportreferenz */
 .
 .
/* Ereignisgesteuerte Verarbeitung:
 * t_event() synchron (T_WAIT) wartend */

for (;;) {
        switch (event = t_event(&tref, T_WAIT, NULL)) {
        .
        .
        /* Daten empfangen; e_bufl ist die TIDU-Länge (t_info()) */

        case T_DATAIN:
            if ((rc = t_datain(&tref,e_bufpt,&e_bufl,&chain)) == T_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei t_datain tref 0x%x\n",
                        t_error(), tref);
                exit (ERROR);
             }
             :
        }
}
/* Daten senden; s_bufl ist höchstens TIDU-Länge */

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

7.3 Senden und Empfangen von Vorrangdaten

Wenn beim Verbindungsaufbau (siehe Abschnitt "Verbindung Aufbauen") der Austausch von Vorrangdaten vereinbart wurde, können die TS-Anwendungen diesen wie folgt vornehmen.

Vorrangdaten sendet man mit dem Aufruf *t_xdatrq()*. Im einfachsten Fall ist der Ablauf so:

– Die sendende TS-Anwendung schickt mit einem Aufruf Vorrangdaten.
– Die empfangende TS-Anwendung erhält das Ereignis T_XDATIN angezeigt. Das ist die Mitteilung, daß Vorrangdaten angekommen sind.
– Die empfangende TS-Anwendung muß die Daten mit dem Aufruf *t_xdatin()* annehmen.

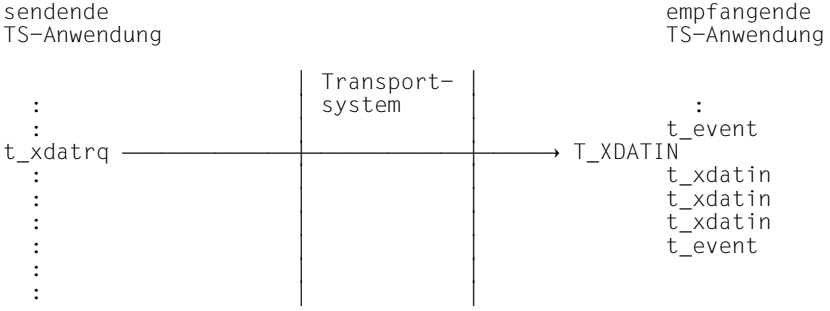


Vorrangdaten übertragen

Der Ergebniswert von t_xdatin()

Bei *t_xdatin()* ist eine Länge anzugeben, in der die angekommenen Vorrangdaten gelesen werden sollen. Wenn die angegebene Länge kleiner ist als die Anzahl der angekommenen Vorrangdaten, gibt der Ergebniswert von *t_xdatin()* die Restlänge der anstehenden Vorrangdaten an.

Sind die Vorrangdaten noch nicht vollständig gelesen, muß erneut *t_xdatin()* aufgerufen werden, und zwar so oft, bis die Daten vollständig gelesen sind. Währenddessen darf auch nicht *t_event()* aufgerufen, die Verbindung umgelenkt oder der Datenfluß geregelt werden.



Vorrangdaten stückweise lesen

7.4 Flußregelung von Daten und Vorrangdaten

Wenn die TS-Anwendung nicht bereit ist, auf einer Verbindung Daten zu empfangen, dann teilt sie dies CMX(BS2000) mit dem Aufruf *t_datastop()* mit. CMX(BS2000) stellt ab sofort das Ereignis T_DATAIN für diese Verbindung nicht mehr zu. Der Kommunikationspartner erhält von CMX(BS2000) bei einem der folgenden *t_datarq()*-Aufrufe das Ergebnis T_DATASTOP und darf keine weiteren Daten senden.

Sobald die TS-Anwendung wieder bereit ist, auf der Verbindung Daten zu empfangen, ruft sie *t_datago()* auf. Die TS-Anwendung kann wieder Daten vom Kommunikationspartner empfangen. Sie erhält das Ereignis T_DATAIN wieder zugestellt.

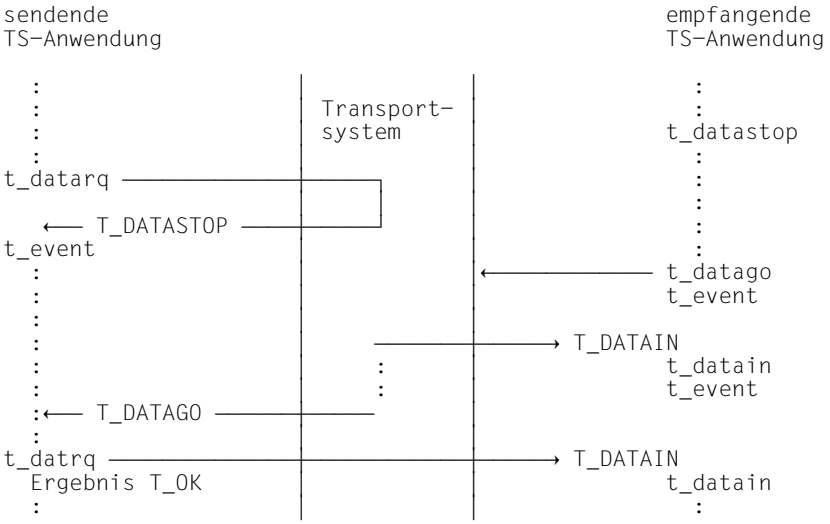
In gleicher Weise erfolgt die Flußregelung für Vorrangdaten. Man verwendet dafür die Aufrufe *t_xdatstop()* und *t_xdatgo()*. Die entsprechenden Ereignisse dazu sind T_XDATIN und T_XDATGO.
Zu beachten ist aber:

Wenn man den Vorrangdatenfluß stoppt (mit *t_xdatstop()*), stoppt CMX(BS2000) implizit auch den Fluß für Normaldaten. Wenn man dann den Vorrangdatenfluß wieder freigibt (mit *t_xdatago()*) bleibt der Fluß für Normaldaten gesperrt! Man muß ihn eigens freigeben (mit *t_datago()*).

Bei Freigabe des Flusses der Normaldaten gibt CMX(BS2000) implizit auch den Fluß der Vorrangdaten wieder frei. Somit gibt man nach einem Aufruf *t_xdatstop()* mit *t_datago()* sowohl den Fluß der Normal- als auch der Vorrangdaten frei.

Welche Vorteile erzielt man, wenn man sich T_DATAIN oder T_XDATIN nicht mehr zustellen läßt?

Die TS-Anwendung kann inzwischen andere CMX(BS2000)-Funktionen aufrufen, z.B. eine weitere Verbindung aufbauen. Das wäre nicht möglich, wenn ein Ereignis T_DATAIN ansteht. Wenn die TS-Anwendung die Daten nicht abholt, meldet jeder Aufruf *t_event()* wieder das Ereignis T_DATAIN und die TS-Anwendung könnte das zum Verbindungsaufbau erforderliche Ereignis T_CONCF nicht erhalten.



Datenflußregelung beim Sender

Die sendende TS-Anwendung erhält T_DATASTOP als Ergebnis des Aufrufes *t_datarq()* oder *t_vdatarq()*, weil die empfangende TS-Anwendung den Datenfluß gestoppt hat oder weil in CMX(BS2000) bzw. BCAM ein zeitweiser Betriebsmittelengpaß vorliegt. Die Daten wurden noch gesendet, aber bei der empfangenden TS-Anwendung nicht mehr angezeigt. Die sendende TS-Anwendung muß nun mit *t_event()* auf das Ereignis T_DATAGO warten, um erneut Daten senden zu können.

8 Die Programmschnittstelle ICMX

Dieses Kapitel beschreibt die Programmschnittstelle ICMX zur Kommunikationsmethode CMX(BS2000). Es enthält:

– einen Überblick über die Funktionen der Schnittstelle ICMX mit Details zu den Kommunikationsphasen,

– Hinweise zur korrekten Verwendung der Funktionen (Zustandsautomaten),

– Hinweise zur Verfügbarkeit der Systemoptionen für die Transportsysteme

– die präzise Beschreibung der Funktionsaufrufe von ICMX mit allen Parametern in alphabetischer Reihenfolge.

8.1 Überblick über die Programmschnittstelle

Transport Service ISO 8072

CMX(BS2000) bietet in der vorliegenden Version mit ICMX eine Programmschnittstelle zum verbindungsorientierten Transport Service (TS) gemäß ISO 8072 im Rahmen des ISO-Referenzmodells für offene Systeme. Daher sind in ICMX die Dienste T-CONNECT (Verbindungsaufbau), T-DISCONNECT (Verbindungsabbau), T-DATA (Datenaustausch), T-EXPEDITED-DATA (Vorrangdatenaustausch) definiert mit den Primitiven:

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

Daneben bietet ICMX lokale Dienste, die die Implementierung von TS-Anwendungen vereinfachen. Dies sind:

T-ATTACH	Anmelden einer TS-Anwendung bei CMX(BS2000)
T-DETACH	Abmelden einer TS-Anwendung bei CMX(BS2000)
T-ERROR	Fehlerabfrage
T-REDIRECT	Umlenken einer Verbindung an einen anderen Task
T-FLOWCONTROL	Flußregelung bei Normaldaten
T-EXPEDITED-FLOWCONTROL	Flußregelung bei Vorrangdaten
T-EVENT	TS-Ereignisabfrage
T-INFO	Information

Der TS erlaubt es zwei TS-Anwendungen, Nachrichten über eine Transportverbingung (TV) auszutauschen. Die verbindungsorientierte Kommunikation bietet den verlust- und duplikatfreien Austausch von Nachrichten unter Beibehaltung der Nachrichtenreihenfolge. Ferner ermöglicht der verbindungsorientierte TS über Verbindungsidentifikationen den Verzicht auf Adreßübertragung und -verarbeitung in der Datenphase.

Eine aufgebaute TV ist (in beiden Endsystemen) durch (je) eine Transportreferenz (tref) zwischen CMX(BS2000) und TS-Anwendung eindeutig identifiziert. Gewisse Parameter, die den Transport der Nachrichten auf der TV beeinflussen, können von den TS-Anwendungen beim Verbindungsaufbau verhandelt werden. Damit die Kommunikation korrekt verläuft, muß man gewisse Regeln beachten, die im folgenden beschrieben werden.

ICMX ist realisiert als eine Menge von C-Funktionen, die die Kommunikation der TS-Anwendungen unabhängig von der speziellen Ausprägung der verwendeten Transportsysteme (die Schichten 1 - 4 im OSI-Referenzmodell) hinsichtlich der Profile, Protokollklassen, etc. machen.

Namen und Adressen

Jede TS-Anwendung hat einen GLOBALEN NAMEN. Unter diesem Namen ist die TS-Anwendung im Netz eindeutig identifizierbar.

Die TS-Anwendung arbeitet ausschließlich mit GLOBALEN NAMEN. Aus ihrem GLOBALEN NAMEN ermittelt eine TS-Anwendung mit Hilfe von CMX(BS2000)-Aufrufen weitere Informationen, z.B. ihren LOKALEN NAMEN, den sie bei der Anmeldung bei CMX(BS2000) angeben muß. Aus dem GLOBALEN NAMEN der entfernten TS-Anwendung ermittelt eine TS-Anwendung die TRANSPORTADRESSE, die sie beim Verbindungsaufbau an CMX(BS2000) übergeben muß.

Durch den LOKALEN NAMEN wird die lokale TS-Anwendung an einen Dienstzugriffspunkt zum TS (TSAP = Transport Service Access Point) gebunden. Die TRANSPORTADRESSE der entfernten TS-Anwendung wird zur Adressierung des Dienstzugriffspunktes im Partnersystem (bzw. der daran gebundenen TS-Anwendung) benötigt.

LOKALER NAME und TRANSPORTADRESSE werden aus dem Transport Name Service gelesen.

ICMX-Funktionen zur Abfrage von Informationen aus dem Transport Name Service sind:

t_getaddr()
liefert zum angegebenen GLOBALEN NAMEN der TS-Anwendung deren TRANSPORTADRESSE. Die TRANSPORTADRESSE muß als Parameter an den entsprechenden Aufruf von ICMX weitergegeben werden.

t_getname()
liefert zur angegebenen TRANSPORTADRESSE den GLOBALEN NAMEN der TS-Anwendung.

t_getloc()
liefert zum angegebenen GLOBALEN NAMEN einer TS-Anwendung deren LOKALEN NAMEN im vorliegenden Endsystem. Der LOKALE NAME muß als Parameter an den entsprechenden Aufruf von ICMX weitergegeben werden.

In <*cmx.h*> sind die Strukturen *t_myname* und *t_partaddr* definiert. In *t_myname* übernimmt (übergibt) die TS-Anwendung ihren LOKALEN NAMEN bei *t_getloc(), t_attach(), t_detach(), t_conrq()* von/an CMX(BS2000) in *t_partaddr* bei *t_getaddr(), t_getname(), t_conin(), t_conrq()* die TRANSPORTADRESSE.

Fehlerbehandlung und -diagnose

Alle Funktionsaufrufe enden mit einer Rückmeldung. Diese zeigt entweder mit T_OK den erfolgreichen Abschluß an, oder informiert durch T_ERROR pauschal über einen aufgetretenen Fehler. Die Fehlerabfragefunktion *t_error()*, sofort nach Auftreten eines Fehlers aufgerufen, liefert detailliertere Diagnoseinformation. Alle Fehlermeldungen, die von CMX(BS2000) als Nichteinhaltung der Kommunikationsregeln durch die TS-Anwendung erkannt werden, haben einen spezifischen Fehlercode und sind in <*cmx.h*> definiert. Die verwendeten Transportsysteme erzeugen keine Fehlermeldungen, eventuelle Fehlerfälle führen zum Verbindungsabbau mit einem entsprechenden Abbaugrund. Den Abbaugrund erhält die TS-Anwendung beim Aufruf von *t_disin()*.

Folgende Funktionen liefern den Klartext zu dem von *t_error()* gelieferten Fehlercode:

t_strerror()
liefert zu einem von ICMX erhaltenen Fehlercode den Zeiger auf den Klartextstring.

t_perror()
ermittelt zu einem von ICMX erhaltenen Fehlercode den Klartextstring mit *t_strerror()* und schreibt ihn nach stderr.

Folgende Funktionen liefern den Klartext zu dem von *t_disin()* gelieferten Abbaugrund:

t_strreason()
liefert zu einem erhaltenen Grund eines Verbindungsabbaus den Zeiger auf den Klartextstring. Der Grund für einen Verbindungsabbau wird der TS-Anwendung beim Aufruf *t_disin()* übergeben.

t_preason()
ermittelt zu einem beim *disin()* erhaltenen Grund eines Verbindungsabbaus den Klartextstring mit *t_strreason()* und schreibt ihn nach stderr.

TS-Anwendungen, Transportverbindungen und Tasks

Eine TS-Anwendung ist ein System von Programmen, das den TS, also die Dienste von CMX(BS2000) "anwendet". Die Abbildung der TS-Anwendung auf das Taskkonzept des Systems bleibt dem Implementierer überlassen. Eine TS-Anwendung kann sich in einem oder mehreren (nicht notwendig verwandten) Tasks organisieren. Die Tasks können prinzipiell unabhängig voneinander TVen zu fernen TS-Anwendungen unterhalten. Die Tasks einer TS-Anwendung können ihre TVen untereinander austauschen. Die Transportreferenz einer TV ist jedoch zu jedem Zeitpunkt genau einer Task zugeordnet. Es gibt in CMX(BS2000) einen eigenen lokalen Dienst REDIRECT zur Umlenkung einer TV an eine andere Task.

Eine Task kann auch gleichzeitig mehrere TS-Anwendungen steuern. In diesem Fall muß bei der Implementierung eine geeignete Koordination der Abläufe in den verschiedenen TS-Anwendungen berücksichtigt werden. CMX(BS2000) unterstützt dies durch den asynchronen Verarbeitungsmodus.

Synchronität und Asynchronität, TS-Ereignisse

Kommunikationsvorgänge sind von Natur aus asynchron: verschiedenste TS-Ereignisse können unabhängig vom Verhalten einer TS-Anwendung auftreten. Zum Beispiel kann eine TS-Anwendung auf einer TV Daten senden, während asynchron für eine andere TV die Anzeige des Verbindungsabbaus eintrifft, worüber die TS-Anwendung unverzüglich informiert werden muß.

Die Funktionen von CMX(BS2000) sind prinzipiell asynchron ausgelegt: das heißt, nach Absetzen eines Aufrufes muß die TS-Anwendung nicht auf die eventuelle Antwort aus dem Netz warten. Diese wird beim Eintreffen von CMX(BS2000) angenommen und der TS-Anwendung bei nächster Gelegenheit auf Anfrage als TS-Ereignis zugestellt.

CMX(BS2000) bietet der TS-Anwendung dazu einen Abfragemechanismus in zwei Ausführungen: synchron (wartend) oder asynchron (prüfend). Diesen Abfragemechanismus muß die TS-Anwendung geeignet verwenden, wenn sie schnell und gezielt auf TS-Ereignisse reagieren will.

Bei synchroner Ausführung wird die aufrufende Task suspendiert, bis ein TS-Ereignis eintrifft. Dieses weckt die Task, damit sie das TS-Ereignis sofort verarbeiten kann. Der Wartezustand kann durch Angabe einer Wartezeit zeitlich begrenzt werden oder durch Aufruf der Funktion *t_wake* - Programm aus *t_event*-Wartezustand wecken - vorzeitig abgebrochen werden. Der synchrone Mechanismus ist nützlich für TS-Anwendungen, die gleichzeitig mehrere TVen unterhalten, damit sie diese nicht pollen müssen.

Bei asynchroner Ausführung kann die Task zu ihr genehmen Zeitpunkten, etwa am Ende eines Verarbeitungsschrittes, nachfragen, ob ein TS-Ereignis eingetreten ist und dieses behandeln, bevor sie mit dem nächsten Verarbeitungsschritt fortfährt. Dies ist nützlich für Tasks, die zwischen zwei TS-Ereignissen längere Pausen erwarten, in denen sie andere Verarbeitungen erledigen können oder müssen.

Die entsprechende Funktion in CMX(BS2000) ist

t_event()
: Sie suspendiert die Task bei Übergabe des Parameterwertes T_WAIT bis zum Eintreten eines TS-Ereignisses, dem Ablaufen der Wartezeit oder dem Aufruf der Funktion *t_wake*. Liegt bereits ein TS-Ereignis oder Fehler vor, kehrt sie sofort mit dessen Code oder T_ERROR als Rückmeldung zurück. Im Unterschied zu CMX(SINIX) wird *t_event()* nicht automatisch bei Ablauf einer Signalroutine beendet. Der Wartezustand wird bei einem Aufruf der Funktion *t_wake* aus einer Signalroutine (Contingency) oder einer anderen Task beendet. *t_event()* kehrt mit T_NOEVENT zurück. Bei Ablauf der Wartezeit setzt sich die Task mit dem TS-Ereignis T_NOEVENT fort. Bei T_CHECK kehrt diese Funktion immer sofort zurück und liefert entweder den Code des gefundenen TS-Ereignisses, oder T_NOEVENT oder T_ERROR.

Folgende asynchrone TS-Ereignisse sind in CMX(BS2000) definiert:

T_NOEVENT
: im asynchronen Fall: kein TS-Ereignis vorhanden,
im synchronen Fall: Abbruch durch Signal oder Ablauf der Wartezeit;

T_CONIN
: Eintreffen einer Verbindungsaufbauanzeige von einer rufenden TS-Anwendung;

T_CONCF
: Eintreffen einer Verbindungsbestätigung von einer gerufenen TS-Anwendung;

T_DISIN
: Eintreffen einer Verbindungsabbauanzeige von einer fernen TS-Anwendung oder von CMX(BS2000);

T_REDIN
: Eintreffen einer Verbindungsumlenkunganzeige von einem anderen Task derselben TS-Anwendung (dieses TS-Ereignis ist lokal, es ist eine Erweiterung des TS zur Flexibilisierung der Implementierung von TS-Anwendungen);

T_DATAIN
: Eintreffen von Normaldaten von einer fernen TS-Anwendung;

T_XDATIN
: Eintreffen von Vorrangdaten von einer fernen TS-Anwendung;

T_DATAGO
: Aufheben einer durch die Flußregelung gesetzten Sendesperre für Normaldaten und Vorrangdaten;

T_XDATGO
: Aufheben einer durch die Flußregelung gesetzten Sendesperre für Vorrangdaten;

T_SYS_EVENT
: *t_event()* konnte ein Signal aus der CMX(BS2000)-Börse nicht als CMX(BS2000)-Ereignis identifizieren;

T_ERROR
: fataler Fehler, nähere Information liefert die Abfragefunktion *t_error()*.

Mit jedem TS-Ereignis, außer T_NOEVENT und T_ERROR, wird der TS-Anwendung auch die Transportreferenz mitgeteilt, damit sie für diese TV spezifisch auf das TS-Ereignis reagieren kann.

Einige TS-Ereignisse müssen von der TS-Anwendung durch Aufrufen entsprechender Funktionen entgegengenommen werden. Ausnahmen sind: T_ERROR, T_DATAGO, T_XDATGO. Diese Funktionsaufrufe liefern weitere Informationen zu den TS-Ereignissen. In der folgenden Tabelle sind die TS-Ereignisse und die zugehörigen Funktionen aufgelistet.

TS-Ereignis	**abholende Funktion**
T_CONCF	t_concf()
T_CONIN	t_conin()
T_DATAIN	t_datain() oder t_vdatain()
T_DISIN	t_disin()
T_REDIN	t_redin()
T_XDATIN	t_xdatin()

TS-Ereignisse werden in der Regel in der Reihenfolge ihres Auftretens zugestellt. Allerdings kann das TS-Ereignis T_XDATIN das TS-Ereignis T_DATAIN überholen, T_DISIN kann T_DATAIN und T_XDATIN überholen. Im letzteren Falle werden die überholten TS-Ereignisse auf dieser TV vernichtet.

Anmeldung/Abmeldung

Die Kommunikation einer Task über CMX(BS2000) wird aktiviert, indem sich die Task bei CMX(BS2000) anmeldet. Durch die erste Task, die sich für eine TS-Anwendung anmeldet, wird diese TS-Anwendung erzeugt. Dabei wird ein Dienstzugriffspunkt (Transport Service Access Point TSAP) eingerichtet, an dem der TS zur Verfügung steht. Mit der Anmeldung der ersten Task wird die TS-Anwendung an diesen TSAP gebunden. Dem TSAP wird der LOKALE NAME der TS-Anwendung zugeordnet. Sie wird damit aus dem Netz adressierbar. Bei der Abmeldung werden noch bestehende TVen und der TSAP abgebaut, die Taskumgebung aufgelöst und belegte Betriebsmittel für zukünftige Verwendung freigegeben.

Dieselbe Task kann sich gleichzeitig für mehrere TS-Anwendungen anmelden (d.h. mehrere TSAP verwalten) und in jeder dieser TS-Anwendungen mehrere Verbindungsendpunkte (Transport Connection Endpoint TCEP) unterhalten. Es können sich auch mehrere Tasks in derselben TS-Anwendung anmelden (denselben TSAP verwenden) und in jeder aktiv TVen aufbauen oder passiv auf Verbindungsanzeigen warten ohne miteinander zu interferieren. Allerdings ist jeder TCEP genau einer Task zugeordnet.

Die folgenden Funktionen dienen zur An- und Abmeldung. Sie erfüllen hauptsächlich lokale Aufgaben. Sofern kein impliziter Verbindungsabbau durchgeführt werden muß, wird keine Information an das Netz übergeben.

t_attach()
: meldet (die laufende Task) einer TS-Anwendung bei CMX(BS2000)/BCAM an. Die Task kann bei der Anmeldung ihr künftiges Verhalten in dieser TS-Anwendung spezifizieren. Mit der ersten Anmeldung beginnt CMX(BS2000), für diese TS-Anwendung Verbindungsaufbauanzeigen entgegenzunehmen;

t_detach()
: meldet (die laufende Task) einer TS-Anwendung bei CMX(BS2000) ab. Noch bestehende TVen der Task in dieser TS-Anwendung baut CMX(BS2000) ab. Sofern keine weitere Task dieser TS-Anwendung angemeldet ist, ist die TS-Anwendung danach bei CMX(BS2000) nicht mehr bekannt.

Verbindungsaufbau, -abbau und -umlenkung

Bevor zwei TS-Anwendungen miteinander Daten austauschen können, muß zwischen ihnen eine TV aufgebaut werden. Eine der beiden TS-Anwendungen wird als die rufende TS-Anwendung angesehen, sie initiiert den Verbindungsaufbau. Die andere ist die gerufene TS-Anwendung, sie wartet auf Anforderungen von rufenden TS-Anwendungen.

Die rufende TS-Anwendung stellt eine Verbindungsanforderung und erhält eine Antwort von der gerufenen TS-Anwendung. Die gerufene TS-Anwendung wartet auf eine Verbindungsanzeige (Anzeige einer Verbindungsanforderung) und nimmt sie an oder weist sie zurück. Während des Verbindungsaufbaus verhandeln die TS-Anwendungen gewisse Merkmale der TV für den Datentransfer und können Benutzerdaten austauschen.

Jede der TS-Anwendungen sowie CMX(BS2000) können jederzeit die TV abbauen. Dies wird von den TS-Anwendungen nicht verhandelt, sondern von CMX(BS2000) sofort vollzogen. Der anderen TS-Anwendung (oder beiden, wenn CMX(BS2000) die TV abbaut) wird eine Verbindungsabbauanzeige zugestellt, die weder beantwortet noch abgewehrt werden kann. Alle Fehlerfälle in den Transportsystemen werden von CMX(BS2000) durch einen Abbau der betroffenen TVen angezeigt. CMX(BS2000) garantiert nicht, daß Daten, die zum Zeitpunkt der Anforderung des Abbaus noch unterwegs sind, noch zugestellt werden.

Die Verbindungsumlenkung ist ein lokaler Dienst in CMX(BS2000), der die Organisation der TS-Anwendung in Tasks vereinfacht. Eine Task, die eine fertig aufgebaute TV hält, kann diese (allerdings abhängig vom Zustand, siehe Bild 8-1) an eine andere Task derselben TS-Anwendung umlenken. Der TSAP und der TCEP bleiben dabei unverändert. Die abgebende Task verliert dabei die Transportreferenz dieser TV, womit diese für sie nicht mehr verfügbar ist. Näheres hierzu im Bild "Zustände von TS-Anwendungen und erlaubte Übergänge" im nachfolgenden Kapitel "Zustände von TS-Anwendungen und Zustandsübergänge".

Die entsprechenden Funktionen sind:

t_conrq()
: fordert den Verbindungsaufbau zur gerufenen TS-Anwendung mit der angegebenen TRANSPORTADRESSE. Der Bezug zum TSAP wird durch den bei der Anmeldung verwendeten LOKALEN NAMEN hergestellt. Die Funktion kehrt nach Absetzen der Anforderung sofort zurück, die rufende TS-Anwendung erhält eine Transportreferenz. Sie muß dann synchron oder asynchron auf die Antwort der gerufenen TS-Anwendung warten (siehe oben).

t_concf()
: übernimmt von CMX(BS2000) die mit T_CONCF angezeigte Antwort der gerufenen TS-Anwendung; damit ist der Verbindungsaufbau vollzogen;

t_conin()
: übernimmt von CMX(BS2000) die mit T_CONIN angezeigte Verbindungsanforderung der rufenden TS-Anwendung mit deren TRANSPORTADRESSE. Der Bezug zum TSAP wird für die gerufene TS-Anwendung durch Lieferung des bei der der Anmeldung angegebenen LOKALEN NAMENS hergestellt;

t_conrs()
: beantwortet (akzeptiert) die Verbindungsanforderung, nachdem sie mit T_CONIN angezeigt und von der TS-Anwendung übernommen wurde;

t_disrq()
: fordert den Verbindungsabbau an; die Funktion kann jederzeit von jeder der TS-Anwendungen aufgerufen werden; mit ihr wird auch eine Verbindungsaufbauanforderung abgelehnt (statt akzeptiert), nachdem sie durch CMX(BS2000) angezeigt und von der TS-Anwendung übernommen wurde;

t_disin()
übernimmt von CMX(BS2000) die mit T_DISIN angezeigte Verbindungsabbauanzeige; durch diesen Funktionsaufruf wird der TS-Anwendung auch der Grund für den Verbindungsabbau übergeben;

t_redrq()
lenkt die TV an eine Task derselben TS-Anwendung um, für die abgebende Task ist sie dann nicht mehr verfügbar;

t_redin()
übernimmt von CMX(BS2000) die mit T_REDIN angezeigte Verbindungsumlenkung; die empfangende Task muß sie annehmen, kann sie aber sofort weitergeben (zurückgeben) oder abbauen.

Datenaustausch und Flußregelung

Sobald eine Verbindung aufgebaut ist, liegt die Initiative bei der TS-Anwendung (nicht bei CMX(BS2000)). Sie kann:

– Normaldaten und (falls vereinbart) Vorrangdaten senden oder

– durch *t_event()* anzeigen, daß sie bereit ist, Normaldaten bzw. (falls vereinbart) Vorrangdaten zu empfangen.

Der Datentransfer findet nachrichten-orientiert statt: die TS-Anwendungen tauschen Transport Service Data Units (TSDU) - Nachrichten beliebiger Länge - oder Expedited Transport Service Data Units (ETSDU) - Vorrangdaten beschränkter Länge - aus. Vorrangdaten sind auf wenige Bytes beschränkt, sie werden mit Vorrang zum Strom der Normaldaten übertragen und in eigenen Warteschlangen geführt. CMX(BS2000) garantiert nur, daß Vorrangdaten nie später bei der empfangenden TS-Anwendung eintreffen als die Normaldaten, die nach ihnen abgesendet wurden. Pro Aufruf kann an CMX(BS2000) höchstens eine komplette ETSDU übergeben werden.

Die Übergabe einer (prinzipiell beliebig langen) TSDU an CMX(BS2000) erfolgt in Portionen der Länge einer Transport Interface Data Unit (TIDU). Die maximale Länge der TIDU ist TV-spezifisch. Sie muß deshalb für jede TV von CMX(BS2000) abgefragt werden (*t_info()*). Die TSDU muß also eventuell mit mehreren Sendeaufrufen übertragen werden. Ein Parameter beim Sendeaufruf zeigt an, ob eine weitere TIDU für diese TSDU folgt (T_MORE) oder nicht (T_END). Daraus ist nicht ableitbar, wie die TIDU für die Übertragung oder Zustellung zur empfangenden TS-Anwendung verpackt wird. CMX(BS2000) garantiert nur, daß die sequentielle Zusammenfügung der TIDUs auf Empfangsseite wieder die TSDU der Sendeseite liefert. Die maximale TIDU-Länge kann für beide TS-Anwendungen verschieden sein und hängt von der TV ab. CMX(BS2000) garantiert nicht, daß bei der empfangenden TS-Anwendung eine TIDU einer TSDU maximal gefüllt zugestellt wird.

Die empfangende TS-Anwendung erhält die Ankunft einer TIDU aus einer TSDU (die Ankunft einer ETSDU) über das TS-Ereignis T_DATAIN (T_XDATIN) angezeigt. Sie holt dann die TIDU (ETSDU) mit einem entsprechenden Funktionsaufruf ganz oder teilweise ab. Gegebenenfalls kann oder muß sie mehrere derartige Aufrufe absetzen, um eine TIDU (ETSDU) von CMX(BS2000) zu übernehmen.

Die Übertragung von TIDUs (ETSDUs) unterliegt Flußregelungsmechanismen, die von CMX(BS2000) und den TS-Anwendungen geregelt werden können. Die Rückmeldung T_DATASTOP (T_XDATSTOP) beim Senden sagt der sendenden TS-Anwendung, daß die TIDU (ETSDU) zwar verarbeitet ist, der Fluß der TIDUs (ETSDUs) aber gesperrt wurde. Es kann keine TIDU (ETSDU) mehr gesendet werden, bis der Fluß wieder freigegeben wird. Dies wird durch das TS-Ereignis T_DATAGO (T_XDATGO) angezeigt.

Die empfangende TS-Anwendung sperrt und entsperrt den Fluß der TIDUs (ETSDUs) durch Funktionsaufrufe an CMX(BS2000), die sich für die sendende TS-Anwendung wie oben auswirken.

Die folgenden Funktionen realisieren Datenaustausch und (aktive) Flußregelung:

t_datarq()
: fordert die Übertragung einer TIDU aus einem zusammenhängenden Speicherbereich an; die Rückmeldung T_DATASTOP besagt, daß der Fluß gesperrt ist; weitere Sendeanforderungen werden mit Fehler abgewiesen, bis der Fluß wieder freigegeben wird;

t_vdatarq()
: wirkt wie *t_datarq*, die TIDU kann aber in mehreren nicht zusammenhängenden Speicherbereichen bereitgestellt werden;

t_datain()
: übernimmt die Daten einer TIDU von CMX(BS2000) in einen zusammenhängenden Speicherbereich, nachdem sie mit T_DATAIN angezeigt wurden; die Rückmeldung gibt an, wieviele Daten noch in der laufenden TIDU enthalten sind; damit kann eine TIDU stückweise gelesen werden;

t_vdatain()
: wirkt wie *t_datain*, kann die TIDU aber in mehrere nicht zusammenhängende Speicherbereiche übernehmen;

t_xdatrq()
: fordert die Übertragung einer ETSDU an; die Rückmeldung T_XDATSTOP besagt, daß der Fluß gesperrt ist; weitere Sendeanforderungen werden dann mit Fehler abgewiesen, bis der Fluß wieder freigegeben wird;

t_xdatin()
: übernimmt die Daten einer ETSDU von CMX(BS2000), nachdem sie mit T_XDATIN angezeigt wurden; die Rückmeldung gibt an, wieviele Daten noch in der laufenden ETSDU enthalten sind; damit kann eine ETSDU stückweise gelesen werden;

t_datastop()
sperrt empfangsseitig den Fluß der Normaldaten auf einer Verbindung; das TS-Ereignis T_DATAIN wird von CMX(BS2000) für diese Verbindung nicht mehr angezeigt;

t_datago()
gibt empfangsseitig den (gesperrten) Fluß der Normaldaten und Vorrangdaten auf einer Verbindung wieder frei; die TS-Ereignisse T_DATAIN und T_XDATIN werden von CMX(BS2000) für diese Verbindung wieder angezeigt;

t_xdatstop()
sperrt empfangsseitig den Fluß der Vorrangdaten und der Normaldaten auf einer Verbindung; CMX(BS2000) zeigt für diese Verbindung die TS-Ereignisse T_XDATIN und T_DATAIN nicht mehr an;

t_xdatgo()
gibt empfangsseitig den (gesperrten) Fluß der Vorrangdaten auf einer Verbindung wieder frei; das Ereignis T_XDATIN wird von CMX(BS2000) für diese Verbindung wieder angezeigt.

t_wake()
weckt eine andere oder die eigene Task aus *t_event()*. Die geweckte Task erhält den Returnwert T_NOEVENT. *t_wake()* ist zum Einsynchronisieren anderer als CMX(BS2000)-Ereignisse in den CMX(BS2000)-Wartepunkt gedacht. In TU können nur Tasks mit derselben USER-ID geweckt werden. *t_wake()* erzeugt unbedingt ein T_NOEVENT-Ereignis, auch wenn die zu weckende Task gerade nicht *t_event()* aufruft.

Informationsdienst

Der Informationsdienst ist ein lokaler Dienst, mit dem die TS-Anwendung konfigurationsabhängige Parameterwerte von CMX(BS2000) abfragen kann. Der Informationsdienst ist durch die folgende Funktion realisiert:

t_info()
liefert für eine eingerichtete TV die Länge der TIDU.

8.2 Zustände von TS-Anwendungen und Zustandsübergänge

Die Abläufe an der Programmschnittstelle ICMX sind in dem folgenden Diagramm durch Zustandsautomaten dargestellt. Das Diagramm zeigt die definierten Zustände, die eine TS-Anwendung im Verlauf der Kommunikation einnehmen kann und die erlaubten Übergänge zwischen diesen Zuständen. Anhand des Diagramms kann man erlaubte Folgen von CMX(BS2000)-Aufrufen ablesen. Es zeigt, wann die Tasks einer TS-Anwendung wie auf bestimmte Ereignisse reagieren sollen.

Programme, die sich in ihrem Verhalten nach diesem Zustandsdiagramm richten, sind kompatibel zu CMX-SINIX; CMX(BS2000) und CMX-SINIX reagieren in diesem Fall ähnlich. Bei abweichendem Verhalten jedoch sind die Ergebnisse und Reaktionen unterschiedlich.

Im Diagramm ist jeder Zustand durch ein doppeltgerahmtes Rechteck dargestellt. Im Rechteck steht der Name des Zustands.

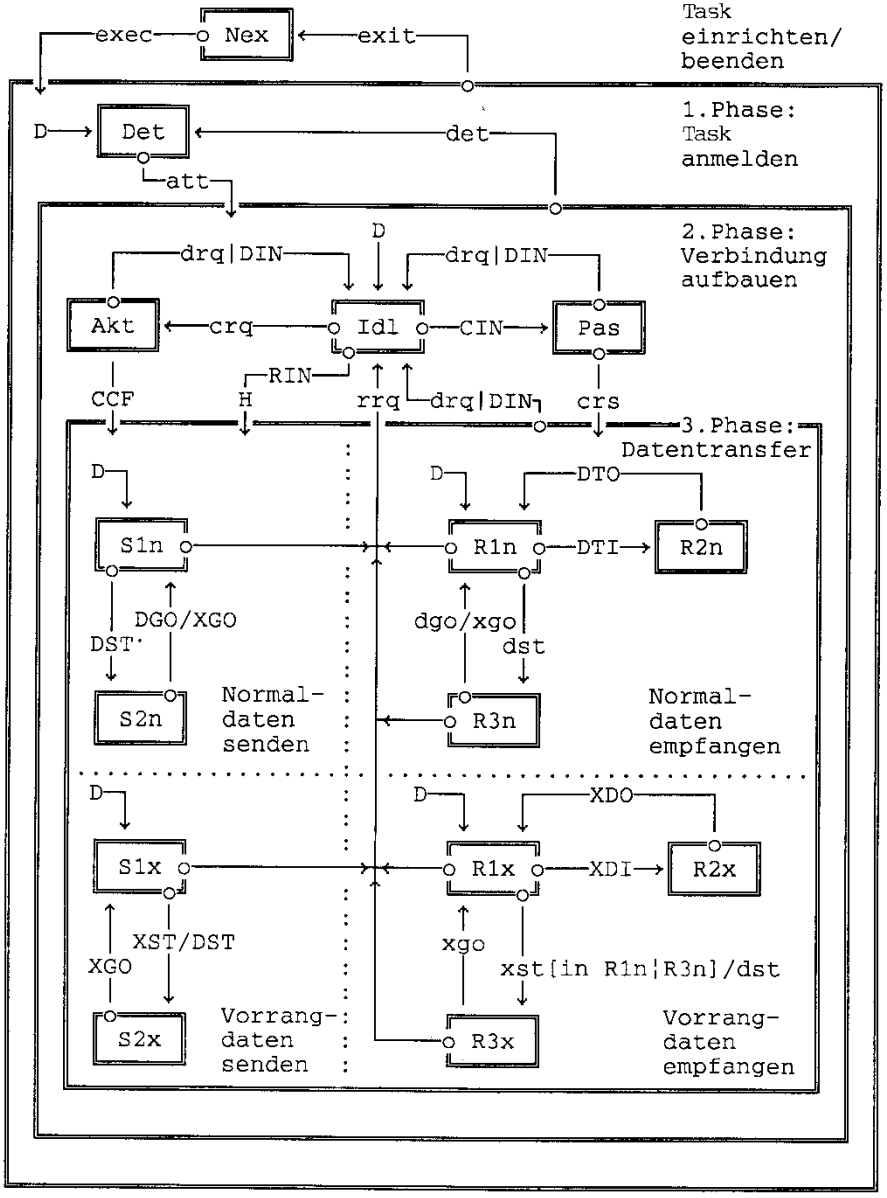
Die umfassenden (äußeren) Rechtecke stellen die drei Phasen der Kommunikation dar.

1. Phase der Kommunikation: Task anmelden
   Die Task existiert, sie ist noch nicht oder nicht mehr bei CMX(BS2000) angemeldet.
2. Phase der Kommunikation: Verbindung aufbauen
   Die Task ist bei CMX(BS2000) angemeldet und es besteht keine Verbindung. Eine Verbindung kann aufgebaut werden.
3. Phase der Kommunikation: Datentransfer
   Die Verbindung ist aufgebaut. Die Task kann Daten senden und empfangen.

Die 3. Phase der Kommunikation ist durch gepunktete Linien in vier Teilbereiche unterteilt. Die Teilbereiche sind:

– Normaldaten senden
– Normaldaten empfangen
– Vorrangdaten senden
– Vorrangdaten empfangen

Die Task befindet sich, wenn sie diese Phase erreicht, zu jeder Zeit in jedem Teilbereich in genau einem Zustand. Es sind nur bestimmte Kombinationen von Zuständen dieser Teilbereiche erlaubt. D.h. ein Zustandsübergang innerhalb eines Teilbereichs kann einen Zustandsübergang in einem anderen Teilbereich bedingen. Wie die einzelnen Zustände in den Teilbereichen verknüpft sind, kann man anhand der Bedingungen für Zustandsübergänge ablesen (siehe unten). Ist der Austausch von Vorrangdaten für die Verbindung nicht vereinbart, so befindet sich die Task nur in Zuständen der oberen beiden Teilbereiche.



Zustände von TS-Anwendungen und erlaubte Übergänge

Die Pfeile o-B→ zwischen den Rechtecken zeigen die möglichen Übergänge. B bezeichnet die Bedingung für den Übergang von Ausgangszustand in den Zielzustand (Ausgangszustand o-B→ Zielzustand). Die Übergänge sind nur in der vom Pfeil angegebenen Richtung möglich.

Im folgenden sind zunächst die Abkürzungen erklärt, die im Diagramm verwendet wurden.

Abkürzungen für die Zustände:

Nex	die Task existiert nicht (mehr)
Det	die TS-Anwendung ist noch nicht bei CMX(BS2000) angemeldet bzw. die TS-Anwendung ist bei CMX(BS2000) abgemeldet.
Idl	Grundzustand zum Verbindungsaufbau und zum Entgegennehmen einer Verbindungsumlenkung bzw. eine vorher bestehende Verbindung wurde abgebaut.
Akt	Warten auf das Ereignis T_CONCF nach dem Aufruf t_conrq() (aktiver Verbindungsaufbau).
Pas	ein Ereignis T_CONIN eingetroffen (passiver Verbindungsaufbau).
S1n	Grundzustand für *t_datarq()* bzw. *t_vdatarq()*.
S2n	Normaldatenfluß gesperrt
R1n	Grundzustand für t_datain()
R2n	T_DATAIN angezeigt
R3n	T_DATAIN gesperrt
S1x	Grundzustand für t_xdatrq()
S2x	Vorrangdatenfluß gesperrt
R1x	Grundzustand für t_xdatin()
R2x	T_XDATIN angezeigt
R3x	T_XDATIN gesperrt

Abkürzungen für die Übergangsbedingungen

exec	Programmstart
exit	Programmende

Folgende Übergänge sind durch Aufruf einer CMX(BS2000)-Funktion bedingt:

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()

Folgende Übergänge sind durch die Übernahme von Ereignissen bedingt:

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

Folgende Übergänge sind durch bestimmte Rückgabewerte der CMX(BS2000)-Funktionen bedingt:

DST	T_DATASTOP bei t_datarq() bzw. t_vdatarq()
XST	T_XDATSTOP bei t_xdatrq()
DTO	0 bei t_datain() bzw. t_vdatain() (aktuelle TIDU vollständig gelesen)
XDO	0 bei t_xdatin() (ETSDU vollständig gelesen)

Erläuterungen zu den möglichen Zustandsübergängen

Pfeile, die an einem umfassenden Rechteck enden, besagen, daß die Task standardmäßig zunächst in die durch D → angezeigten Zustände übergeht. Z.B. beim Übergang in die 3. Phase der Kommunikation (Datentransfer) geht die Task zunächst in die Zustände S1n, S1x, R1n, R1x über.
Eine Ausnahme ist der Übergang RIN−H→. Er bedeutet: Bei der Verbindungsumlenkung nimmt die empfangende Task in der 3. Phase (Datentransfer) die Zustände ein, die die umlenkende Task vor der Verbindungsumlenkung in dieser Phase eingenommen hat.

Pfeile, die an den umfassenden Rechtecken beginnen, besagen, daß der Übergang von einem beliebigen Zustand innerhalb des Rechtecks erfolgen kann.

Zustandsübergänge dieser Art sind:

– exec
  Die Task startet ein Anwendungsprogramm, das CMX-Funktionen verwenden kann.

– exit
  Ein Anwendungsprogramm wird beendet. Alle Verbindungen werden von CMX(BS2000) abgebaut.

– det
  Ruft die Task in einem beliebigen Zustand *t_detach()* auf, so geht sie in den Zustand Det über. CMX(BS2000) baut seine Verbindungen ab.

– drq|DIN (drq oder DIN)
  Ruft die Task in einem beliebigen Zustand beim Datentransfer (3. Phase) oder innerhalb des Verbindungsaufbaus (2. Phase) *t_disrq()* auf, so geht die Task in den Zustand Idl über. Das gleiche geschieht, wenn CMX(BS2000) der Task das Ereignis T_DISIN anzeigt. Die bestehende Verbindung wird abgebaut oder der Verbindungswunsch einer anderen TS-Anwendung abgelehnt.

Zustandsübergänge innerhalb der 3. Phase (Datentransfer)

Im folgenden werden die Verknüpfungen von Zustandsübergängen in den Teilbereichen der 3. Phase beschrieben. Der Zustand, den die Task im Teilbereich "Normaldaten senden" einnimmt, ist abhängig von ihrem Zustand im Teilbereich "Vorrangdaten senden" und umgekehrt. Der Zustand, den die Task im Teilbereich "Normaldaten empfangen" einnimmt, ist abhängig von ihrem Zustand im Teilbereich "Vorrangdaten empfangen" und umgekehrt.

Folgende Verknüpfungen bestehen zwischen den Zuständen der vier Teilbereiche:

DGO/XGO (DGO löst XGO aus)
: Das Ereignis T_DATAGO löst T_XDATGO aus. Mit dem Normaldatenfluß wird Vorrangdatenfluß freigegeben, sofern er gesperrt ist. Somit löst der Übergang S2n → S1n den Übergang S2x → S1x aus.

XST/DST (XST löst DST aus)
: Das Ereignis T_XDATSTOP löst das Ereignis T_DATASTOP aus. Der Übergang S1x → S2x bewirkt den Übergang S1n → S2n. Eine Sperre des Vorrangdatenflusses bedingt die Sperre des Normaldatenflusses.

dgo/xgo (dgo löst xgo aus)
: Ruft die Task im Zustand R3n (T_DATAIN gesperrt) *t_datago()* auf, so wird implizit *t_xdatgo()* aufgerufen. Der Übergang R3n → R1n löst den Übergang R3x → R1x aus, sofern die Task zuvor den Zustand R3x eingenommen hat.

xst[in R1n|R3n]/dst
: Befindet sich die Task im Zustand R1x, so kann sie nur *t_xdatstop()* aufrufen, wenn sie sich im Teilbereich "Normaldaten empfangen" im Zustand R1n oder R3n befindet. Sie löst dabei *t_datastop()* aus. D.h. der Vorrangdatenfluß kann von der Task nur gesperrt werden, solange kein T_DATAIN angezeigt wird. Mit dem Vorrangdatenfluß wird implizit der Normaldatenfluß gesperrt (R1x → R3x löst R1n → R3n aus).

8.3 Systemoptionen und Nachrichtenlänge

Beim Erstellen von TS-Anwendungsprogrammen sollten Sie beachten, daß die Systemoptionen "Benutzerdatenaustausch beim Verbindungsaufbau und -abbau" und "Austausch von Vorrangdaten" nicht von jeder Transportverbindung unterstützt werden. Bei den Transportverbindungen, die diese Systemoptionen unterstützen, ist die erlaubte Länge der Benutzerdaten bzw. der Vorrangdateneinheit verschieden.

8.4 Programmierhinweise

Das Hauptziel von ICMX ist, die TS-Anwendungen von den verwendeten Transportsystemen unabhängig zu machen. Dies versetzt die TS-Anwendungen in die Lage, in unterschiedlichen Netzumgebungen ablaufen zu können. ICMX unterstützt die Unabhängigkeit für solche TS-Anwendungen, die den folgenden Regeln genügen:

1) Die Anwendung sollte keine expliziten Annahmen über die Länge einer TIDU machen oder wie die TIDUs zur Kommunikation verpackt werden.

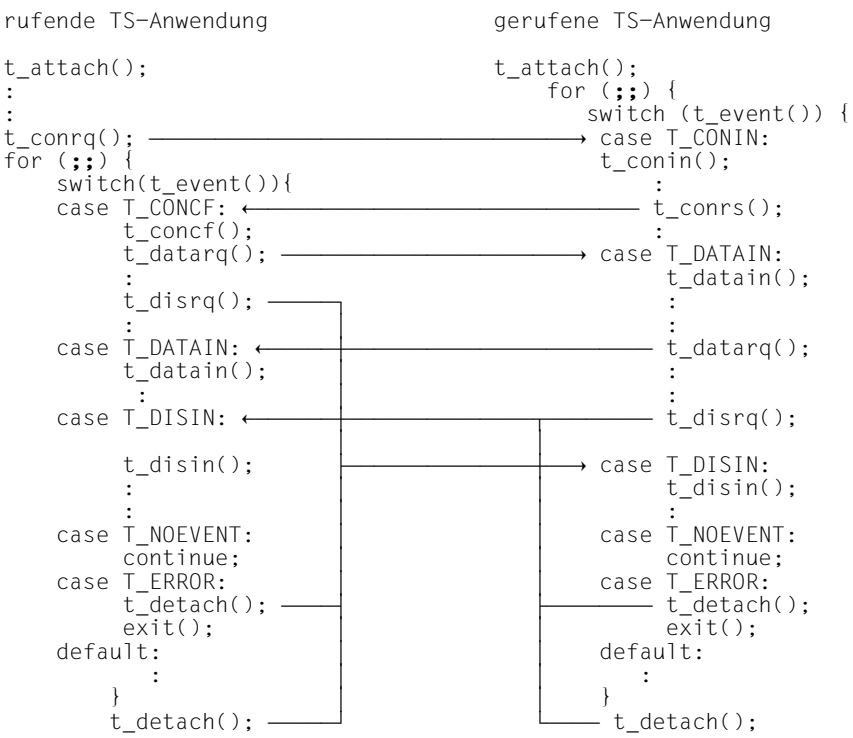
2) Die in <*cmx.h*> festgesetzten Grenzwerte für die Optionen dürfen keinesfalls überschritten werden. Es ist zu beachten, daß manche Transportsysteme gewisse Optionen nicht bieten.

3) Die TS-Anwendung muß die Namensadreßumsetzung ausschließlich mit Hilfe der Funktionen *t_getloc()*, *t_getaddr()* und *t_getname()* durchführen. Grundlage für diese Funktionen sind die Name-Service-Einträge im BCAM-Mapping.

4) CMX(BS2000)-Funktionen sollten nicht in Contingencies aufgerufen werden. Die Contingencies sind nicht dazu geschaffen, asynchrone CMX-Bearbeitung außerhalb des laufenden Kontextes vorzunehmen.

5) Die Programmlogik sollte in einer switch/case-Konstruktion aufgebaut werden, die für diese Zwecke bestens geeignet ist.

Beispiel


rufende TS-Anwendung
gerufene TS-Anwendung
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 Konventionen

Bei Verwendung von ICMX sind folgende Konventionen einzuhalten:

1) alle Identifier, die mit _ (Unterstrich) beginnen, sind reserviert für die Systemsoftware;
2) alle Identifier, die mit t_ oder ts oder Ts beginnen, sind für CMX(BS2000) reserviert;
3) alle Präprozessordefinitionen, die mit T_ oder TS_ beginnen, sind für CMX(BS2000) reserviert;

8.6 ICMX - Funktionsaufrufe

Die folgenden Seiten beschreiben die CMX(BS2000)-Aufrufe im Detail. Kursivdruck im Fließtext repräsentiert gewöhnlich ersetzbare Formalparameter oder die Namen von Funktionen und Dateien. Namen in Großbuchstaben (z.B. T_MSGSIZE) stehen für Konstanten, die in einer Include-Datei durch #define definiert sind.

Folgende Kennzeichnungen werden bei der Parameterbeschreibung verwendet:

-> kennzeichnet Parameter, in denen CMX(BS2000) einen vom Aufrufer bereitgestellten Wert erwartet.

<- kennzeichnet Parameter, in denen CMX(BS2000) nach dem Aufruf einen Wert liefert.

<> kennzeichnet Parameter, in denen der Aufrufer einen Wert bereitstellen muß, der dann von CMX(BS2000) modifiziert wird. Die Modifikation erfolgt im allgemeinen nur bei positivem Ablauf. Bei negativem Verlauf bleibt der Wert unverändert.

Wenn es sich bei dem Parameter um einen Zeiger handelt, bezieht sich das Kennzeichen natürlich nicht auf diesen (wird immer vom Aufrufer bereitgestellt), sondern auf den Inhalt des Feldes, auf das der Zeiger zeigt.

In allen Fällen muß entsprechender Speicherplatz für alle von CMX(BS2000) zu liefernden Werte vom Aufrufer bereitgestellt und ein Zeiger an CMX(BS2000) übergeben werden.

Alle Fehlerwerte, die bei den einzelnen Aufrufen sich im Fehlerfall ergeben können, sind im Anhang aufgeführt.

t_attach
Anmelden einer Task bei CMX(BS2000) (attach task)

t_attach() meldet die laufende Task für die TS-Anwendung bei CMX(BS2000) an. Ist diese Task die erste, die sich für diese TS-Anwendung anmeldet, wird die TS-Anwendung (TSAP) eingerichtet. Weitere Tasks können sich für dieselbe TS-Anwendung anmelden. Dieselbe Task kann sich mit *t_attach()* auch für mehrere TS-Anwendungen anmelden. Dabei ist der LOKALE NAME der TS-Anwendung als Parameter anzugeben.

Im BS2000 beinhaltet der von *t_getloc()* gelieferte LOKALE NAME jedoch nicht die Transportadressen, sondern den syntaktisch etwas veränderten GLOBALEN NAMEN. Die Umwandlung in Transportadressen geschieht erst beim *t_attach()* intern in BCAM mit Hilfe der Einträge im Name-Mapping. Der Name-Server wird durch das BCAM-Mapping ersetzt.

Über BCAM-Mapping ist es möglich, privilegierte TS-Anwendungen zu definieren. Dazu muß per /BCMAP dem GLOBALEN NAMEN der TS-Anwendung ein privilegierter NEA-Tsel (1. Zeichen = "$") und/oder eine privilegierte Portnummer zugewiesen werden. So definierte TS-Anwendungen können nur von Tasks eröffnet werden, die das TSOS- oder NETADM-Privileg besitzen.

Nach der ersten erfolgreichen Anmeldung für eine TS-Anwendung beginnt CMX(BS2000), für diese Anwendung Verbindungswünsche entgegenzunehmen. Die Verbindungswünsche werden immer der ältesten Task, die sich für den TSAP angemeldet hat (i.allg. die erste, die diesen TSAP eröffnet hat), zugestellt.

Durch die Parameter, die beim Aufruf von *t_attach()* übergeben werden, legt man fest, für welche TS-Anwendung sich die Task anmeldet.

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

-> name
    Zeiger zu einem Datenbereich mit dem LOKALEN NAMEN der TS-Anwendung. Den LOKALEN NAMEN liefert der Aufruf *t_getloc()* als Eigenschaft zum GLOBALEN NAMEN der TS-Anwendung.

<> opt

Für den Parameter *opt* ist der Wert NULL oder der Zeiger zu einer Variante mit Benutzeroptionen anzugeben.
Wird *opt* = NULL angegeben, so setzt CMX(BS2000) die angegebenen Standardwerte.
Die Angabe von *t_opta1* wird nur aus Kompatibilität zu SINIX unterstützt - sie wirkt wie die Angabe von NULL, da CMX(BS2000) keinen der darin enthaltenen Parameter auswertet.

Folgende Strukturen sind in <*cmx.h*> definiert:

```
     struct t_opta1 {
->       int t_optnr;    /* Options-Nr. T_OPTA1 */
->       int t_apmode;   /* Task-Mode */
->       int t_conlim;   /* Verbindungsanzahl */
     }

     struct t_opta2 {
->       int t_optnr;    /* Options-Nr. T_OPTA2 */
->       int t_apmode;   /* Task-Mode */
->       int t_conlim;   /* Verbindungsanzahl */
->       int t_uattid;   /* Benutzerreferenz der Anmeldung */
<-       int t_attid;    /* CMX-Referenz der Anmeldung */
<-       int t_ccbits;   /* reserviert */
<-       int t_sptypes;  /* reserviert */
     }
```

t_optnr

Optionsnummer. Anzugeben ist:

T_OPTA1 bei t_opta1
T_OPTA2 bei t_opta2

t_apmode

t_apmode wird von CMX(BS2000) nicht ausgewertet.

Standardwert im BS2000: T_ACTIVE | T_PASSIVE | T_REDIRECT

t_conlim

t_conlim wird von BS2000 nicht ausgewertet. Die Maximalanzahl der Verbindungen wird von BCAM nicht pro Task sondern pro TSAP begrenzt. Dieser Wert kann von der BCAM-Administration BCAM-weit festgelegt werden.

t_uattid

Im Feld *t_uattid* kann man CMX(BS2000) eine beliebige Benutzerreferenz dieser Anmeldung übergeben. Sie wird im folgenden als Option beim *t_event* von CMX(BS2000) zurückgeliefert, d.h. wenn die laufende Task das Eintreffen eines Ereignisses von CMX(BS2000) abfragt.
Mit Hilfe dieser Benutzerreferenz kann eine Task, die mehrere TS-Anwendung steuert, die eintreffenden Ereignisse einfacher den entsprechenden Anmeldungen zuordnen.

Standardwert bei Angabe von NULL: 0

t_attid
: Dieses Feld dient Verfolger- und Diagnosezwecken. Es wird ausschließlich zur Protokollierung verwendet.
Im Feld *t_attid* liefert CMX(BS2000) die CMX(BS2000)-interne Referenz der Anmeldung zurück.

t_ccbits
: Dieses Feld ist reserviert und wird von CMX(BS2000) nicht ausgewertet.

t_sptyps
: Dieses Feld ist reserviert und wird von CMX(BS2000) nicht ausgewertet.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Die Task hat sich als erste mit diesem Namen angemeldet.

T_NOTFIRST
: Der Aufruf war erfolgreich. Die Task hat sich jedoch nicht als erste Task für diese TS-Anwendung angemeldet.

T_ERROR
: Fehler. Fehlercode kann mit *t_error()* abgefragt werden.

Fehler

Fehlerwerte siehe Anhang.

Siehe auch

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

t_concf
Verbindung herstellen (connect confirmation)

t_concf() nimmt ein zuvor mit *t_event()* gemeldetes Ereignis T_CONCF von CMX(BS2000) entgegen. T_CONCF zeigt an, daß die gerufene TS-Anwendung einen Verbindungswunsch der laufenden Task (*t_conrq()*-Aufruf) positiv beantwortet hat.

t_concf() liefert:

– die Benutzerdaten, die die gerufene TS-Anwendung mitgeschickt hat, falls das verwendete Transportsystem diese Option bietet.

– die Antwort der gerufenen TS-Anwendung, wenn die laufende Task beim Verbindungsaufbauwunsch *t_conrq()* den Austausch von Vorrangdaten vorgeschlagen hat.

Wenn der Aufruf *t_concf()* erfolgreich war, ist die Verbindung für die laufende Task aufgebaut. Sobald eine Verbindung aufgebaut ist, liegt die Initiative bei der TS-Anwendung (nicht bei CMX(BS2000)). Sie kann:

– Normaldaten und (falls vereinbart) Vorrangdaten senden oder

– durch *t_event()* anzeigen, daß sie bereit ist, Normaldaten bzw. (falls vereinbart) Vorrangdaten zu empfangen,

– die Verbindung umlenken oder abbauen.

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

-> tref
: Zeiger zu einem Feld mit der Transportreferenz der Verbindung, die der laufenden Task beim *t_event()* übergeben wurde.

<> opt
: Für *opt* ist der Wert NULL oder der Zeiger auf eine Variante anzugeben, die eine Struktur mit Systemoptionen enthält.
Damit werden die Benutzerdaten entgegengenommen, die die gerufene TS-Anwendung bei der Antwort auf den Verbindungswunsch mitgegeben hat.
Wird *opt* = NULL angegeben, so vernichtet CMX(BS2000) die Benutzerdaten. Hat die gerufene TS-Anwendung keine Benutzerdaten und Optionen angegeben, so setzt CMX(BS2000) die angegebenen Standardwerte.

Folgende Struktur ist in <*cmx.h*> definiert:

```
    struct t_optc1 {
->      int  t_optnr;    /* Options-Nr. */
<-      char *t_udatap;  /* Datenpuffer */
< >     int  t_udatal;   /* Länge des Datenpuffers */
<-      int  t_xdata;    /* Vorrangdaten-Auswahl */
<-      int  t_timeout;  /* reserviert */
    };
```

t_optnr
: Optionsnummer. Anzugeben ist T_OPTC1.

t_udatap
: Zeiger auf einen Datenbereich, in den CMX(BS2000) die empfangenen Benutzerdaten der gerufenen TS-Anwendung einträgt. Der Bereich muß so groß sein, daß die empfangenen Daten ganz hineinpassen. Die maximal erforderliche Länge hängt von der verwendeten Transportverbindung ab.

  Standardwert bei Angabe von NULL: undefiniert

t_udatal
: Vor dem Aufruf muß hier 0 oder die Länge des Datenbereiches *t_udatap* stehen. T_CON_SIZE ist die für alle Transportverbindungen geeignete Maximalgröße. T_CON_SIZE ist in <*cmx.h*> definiert.
  Nach dem Aufruf liefert CMX(BS2000) in diesem Feld die Anzahl der Byte zurück, die nach *t_udatap* übertragen wurden.

  Standardwert bei Angabe von NULL: 0.

t_xdata
: CMX(BS2000) liefert hier die Antwort der gerufenen TS-Anwendung, wenn beim Verbindungsaufbau der Austausch von Vorrangdaten vorgeschlagen wurde. Die Antwort ist verbindlich. Dabei bedeutet:

  T_YES
  : die gerufene TS-Anwendung stimmt dem Vorschlag zu,

  T_NO
  : die gerufene TS-Anwendung lehnt den Vorschlag ab.

  Standardwert bei Angabe von NULL: T_NO.

t_timeout
: Der Inhalt dieses Feldes ist stets T_NO.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode kann mit *t_error()* abgefragt werden.

Fehler

Fehlerwerte siehe Anhang.

Siehe auch

t_conrq(), t_error(), t_event()

t_conin
Verbindungswunsch entgegennehmen (connect indication)

t_conin() nimmt ein zuvor mit *t_event()* gemeldetes Ereignis T_CONIN entgegen. T_CONIN zeigt an, daß eine rufende TS-Anwendung eine Verbindung zur laufenden Task aufbauen will.

Der Aufruf liefert:

– die TRANSPORTADRESSE der rufenden TS-Anwendung,
– den LOKALEN NAMEN der lokalen TS-Anwendung,
– die Benutzerdaten, die die rufende TS-Anwendung mitgegeben hat.

Anschließend kann der Verbindungswunsch mit *t_conrs()* beantwortet (bestätigt) oder mit *t_disrq()* abgelehnt werden.

```
#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
: Zeiger zu einem Feld mit der Transportreferenz der Verbindung, die der laufenden Task beim *t_event* übergeben wurde.

<- toaddr
: Zeiger zu einer Variante *t_address*. In dieser Variante liefert CMX(BS2000) den LOKALEN NAMEN der gerufenen TS-Anwendung zurück, die die Verbindung erhalten soll.
Ist die laufende Task in mehreren TS-Anwendungen angemeldet, so kann mit Hilfe dieser Information die Verbindungsanforderung der richtigen TS-Anwendung zugeordnet werden.

<- fromaddr
: Zeiger zu einer Variante *t_address*, in der CMX(BS2000) die TRANSPORTADRESSE der rufenden TS-Anwendung zurückliefert. Die TRANSPORTADRESSE kann mit Hilfe des Aufrufs *t_getname()* in den GLOBALEN NAMEN der rufenden TS-Anwendung übersetzt werden.

<> opt

Für *opt* ist der Wert NULL oder der Zeiger auf eine Variante anzugeben, die eine Struktur mit Systemoptionen enthält.

Damit können die Benutzerdaten abgefragt werden, die die rufende TS-Anwendung beim Verbindungsaufbau angegeben hat.
Wird *opt* = NULL angegeben, so vernichtet CMX(BS2000) die Benutzerdaten. Hat die rufende TS-Anwendung beim *t_conrq()* keine Benutzerdaten und Optionen angegeben, so liefert CMX(BS2000) die angegebenen Standardwerte.

Folgende Struktur ist in <*cmx.h*> definiert:

```
    struct t_optc1 {
->      int  t_optnr;    /* Options-Nr. */
<-      char *t_udatap;  /* Datenpuffer */
< >     int  t_udatal;   /* Länge des Datenpuffers */
<-      int  t_xdata;    /* Vorrangdaten-Auswahl */
<-      int  t_timeout;  /* Inaktiv-Zeit */
    };
```

t_optnr
: Optionsnummer. Anzugeben ist T_OPTC1.

t_udatap
: Zeiger zu einem Datenbereich, in den CMX(BS2000) die empfangenen Benutzerdaten der rufenden TS-Anwendung überträgt. Der Bereich muß so groß sein, daß die empfangenen Daten ganz hineinpassen. Die maximal erforderliche Länge hängt von der verwendeten Transportverbindung ab. T_CON_SIZE ist ist eine für alle Transportverbindungen geeignete Maximalgröße.
T_CON_SIZE ist in <*cmx.h*> definiert.

  Standardwert bei Angabe von NULL: undefiniert

t_udatal
: Vor dem Aufruf muß hier 0 oder die Länge des Datenbereiches *t_udatap* stehen. Nach dem Aufruf liefert CMX(BS2000) in diesem Feld die Anzahl der Byte zurück, die nach *t_udatap* übertragen wurden.

  Standardwert bei Angabe von NULL: 0.

t_xdata
: In diesem Feld liefert CMX(BS2000) den Vorschlag der rufenden TS-Anwendung nach Vorrangdaten zurück.

  Dabei bedeutet:

  T_YES
  : die rufende TS-Anwendung schlägt den Austausch von Vorrangdaten vor.

  T_NO
  : der Austausch von Vorrangdaten wird von der rufenden TS-Anwendung ausgeschlossen.

Schlägt die rufende TS-Anwendung den Austausch von Vorrangdaten vor (T_YES), so ist die Antwort der laufenden Task beim folgenden *t_conrs()* endgültig.
Wünscht die rufende TS-Anwendung keine Vorrangdaten (T_NO), können von der laufenden Task beim folgenden *t_conrs()* auch keine verlangt werden. Gegebenenfalls muß die laufende Task den Verbindungswunsch dann mit *t_disrq()* ablehnen.

Standardwert bei Angabe von NULL: T_NO.

t_timeout
Der Inhalt dieses Feldes ist stets T_NO.

Rückgabewert

T_OK
Der Aufruf war erfolgreich.
T_ERROR
Fehler. Fehlercode kann mit *t_error()* abgefragt werden.

Fehler

Fehlerwerte siehe Anhang.

Siehe auch

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

t_conrq
Verbindung anfordern (connection request)

t_conrq() fordert den Aufbau einer Transportverbindung von der lokalen TS-Anwendung zu einer gerufenen TS-Anwendung an (aktiver Verbindungsaufbau).

Die Verbindungsanforderung *t_conrq()* bewirkt im einzelnen:

– Die gerufene TS-Anwendung erhält das Ereignis T_CONIN als Verbindungsaufbauanzeige. Sie muß darauf antworten.
  Die Antwort der gerufenen TS-Anwendung wird der laufenden Task später beim *t_event()* als Ereignis T_CONCF oder T_DISIN von CMX(BS2000) angezeigt.

– Der gerufenen TS-Anwendung können bei der Verbindungsanforderung Benutzerdaten mitgeschickt werden, falls das verwendete Transportsystem diese Option bietet.

```
#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
: Zeiger zu einem Feld, in dem CMX(BS2000) die verbindungsspezifische Transportreferenz zurückliefert. Sie kennzeichnet die Verbindung in den folgenden Kommunikationsphasen eindeutig. Sie muß deshalb bei allen Aufrufen, die diese Verbindung betreffen, angegeben werden.

-> toaddr
: Zeiger zu einer Variante *t_address* mit der TRANSPORTADRESSE der gerufenen TS-Anwendung. Die TRANSPORTADRESSE liefert der Aufruf *t_getaddr()* als Eigenschaft zum GLOBALEN NAMEN der gerufenen TS-Anwendung. Sie kann zuvor mit Hilfe des Aufrufs *t_getaddr()* ermittelt werden.

-> fromaddr
: Zeiger zu einer Variante *t_address* mit dem LOKALEN NAMEN der rufenden TS-Anwendung. Hier muß derselbe LOKALE NAME angegeben werden wie beim *t_attach()* für diese TS-Anwendung.

-> opt

Für *opt* ist der Wert NULL oder der Zeiger zu einer Variante mit Systemoptionen anzugeben.
Damit werden die Benutzerdaten und Optionen angegeben, die die gerufene TS-Anwendung mit der Verbindungsaufbauanzeige erhalten soll.
Wird *opt* = NULL angegeben, so setzt CMX(BS2000) die angegebenen Standardwerte.

Folgende Struktur ist in <*cmx.h*> definiert:

```
   struct t_optc1 {
->     int  t_optnr;    /* Options-Nr. */
->     char *t_udatap;  /* Datenpuffer */
->     int  t_udatal;   /* Länge des Datenpuffers */
->     int  t_xdata;    /* Vorrangdaten-Auswahl */
->     int  t_timeout;  /* Inaktiv-Zeit */
   };

   struct t_optc3 {
->     int  t_optnr;    /* Options-Nr. */
->     char *t_udatap;  /* Datenpuffer */
->     int  t_udatal;   /* Länge des Datenpuffers */
->     int  t_xdata;    /* Vorrangdaten-Auswahl */
->     int  t_timeout;  /* Inaktiv-Zeit */
->     int  t_ucepid;   /* Benutzerreferenz der Verbindung */
   };
```

t_optnr

Optionsnummer. Anzugeben ist:

T_OPTC1 bei t_optc1
T_OPTC3 bei t_optc3

t_udatap

Zeiger zu einem Bereich mit Benutzerdaten, die die gerufene TS-Anwendung mit der Verbindungsaufbauanzeige erhalten soll.

Standardwert bei Angabe von NULL: undefiniert

t_udatal

Länge der Benutzerdaten in Byte, die aus dem Bereich *t_udatap* zu übertragen sind.
Wird für *t_udatal* 0 angegeben, so wird *t_udatap* nicht ausgewertet.
Der Maximalwert für *t_udatal* ist abhängig von der aufzubauenden Transportverbindung.

Standardwert bei Angabe von NULL: 0

t_xdata
: Im Parameter *t_xdata* teilt die laufende Task der gerufenen TS-Anwendung mit, ob sie bereit ist, Vorrangdaten auszutauschen oder nicht.

  Zulässige Werte sind:

  T_YES
  : Der Austausch von Vorrangdaten wird vorgeschlagen.

  T_NO
  : Der Austausch von Vorrangdaten wird ausgeschlossen.

  Standardwert bei Angabe von NULL: T_NO.

t_timeout
: Ist nur aus Kompatibilitätsgründen zu SINIX in der Struktur definiert. In CMX(BS2000) gibt es keine Zeitüberwachung der Verbindung, der Parameter wird von CMX(BS2000) kommentarlos ignoriert. Standardwert: T_NO.

t_ucepid
: In diesem Feld kann eine beliebige Benutzerreferenz dieser Verbindung an CMX(BS2000) übergeben werden.
  Diese Benutzerreferenz kann der laufenden Task als Option beim *t_event()* von CMX(BS2000) zurückgeliefert werden.
  Damit kann die laufende Task, falls sie mehrere Verbindungen hält, Ereignisse der entsprechenden Verbindung über ein selbst gewähltes Merkmal zuordnen. Die Benutzerreferenz ist eine Alternative zu der von CMX(BS2000) gewählten Transportreferenz *tref*.

  Standardwert bei Angabe von NULL: 0

Hinweise

Stehen dem Transportsystem mehrere Routen (aus der Generierung) für die Transportverbindung zu einer TS-Anwendung zur Verfügung, dann wählt das Transportsystem selbst eine ihm geeignet erscheinende aus. Nur über einen BCAM-MAPPING-Eintrag kann für die gerufene TS-Anwendung auch eine spezielle Route zugeordnet werden - diese wird dann immer benutzt.

Erlaubt es das darunterliegende Protokoll nicht, Verbindungsdaten auszutauschen, dann gehen diese u.U. kommentarlos verloren.

Die Speicherbereiche müssen allokiert und lesend (**fromaddr*, **toaddr*, **opt*, **t_udatap*) bzw. schreibend (**tref*) zugreifbar sein - andernfalls bricht das Programm mit Adreßfehler ab! Daß Benutzerdaten angegeben sind, erkennt CMX(BS2000) an *t_udatal* > 0, *t_udatap* = NULL ist erlaubt.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

t_conrs
Verbindungswunsch beantworten (connection response)

Mit *t_conrs()* akzeptiert (bestätigt) die gerufene TS-Anwendung den Verbindungsaufbauwunsch einer rufenden TS-Anwendung. Der Verbindungsaufbauwunsch wurde der laufenden Task zuvor beim *t_event()* mit dem Ereignis T_CONIN angezeigt. Sie muß das Ereignis T_CONIN vor dem *t_conrs()* mit *t_conin()* entgegennehmen (passiver Verbindungsaufbau). Die rufende TS-Anwendung erhält diese Antwort als Verbindungsbestätigung mit dem Ereignis T_CONCF.

Mit der Antwort *t_conrs()*

– können der rufenden TS-Anwendung Benutzerdaten mitgeschickt werden, falls das verwendete Transportsystem diese Option bietet.

– ist die Verbindung für die laufende Task fertig aufgebaut.

Sobald eine Verbindung etabliert ist, liegt die Initiative bei der TS-Anwendung (nicht bei CMX(BS2000)). Sie kann:

– sowohl Normaldaten als auch (falls vereinbart) Vorrangdaten senden oder

– durch *t_event()* anzeigen, daß sie bereit ist, Normaldaten bzw. (falls vereinbart) Vorrangdaten zu empfangen.

– die Verbindung abbauen oder umlenken.

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

-> tref
Zeiger zu einem Feld mit der Transportreferenz der Verbindung, die beim entsprechenden *t_conin()* verwendet wurde.

-> opt
Für *opt* ist der Wert NULL oder der Zeiger zu einer Variante mit Systemoptionen anzugeben.

Damit übergibt die laufende Task die Benutzerdaten, die die rufende TS-Anwendung mit der Antwort auf den Verbindungswunsch erhalten soll. Wird *opt* = NULL angegeben, so setzt CMX(BS2000) die angegebenen Standardwerte.

Folgende Struktur ist in <*cmx.h*> definiert:

```
    struct t_optc1 {
->      int  t_optnr;    /* Options-Nr. */
->      char *t_udatap;  /* Datenpuffer */
->      int  t_udatal;   /* Länge des Datenpuffers */
->      int  t_xdata;    /* Vorrangdaten-Auswahl */
->      int  t_timeout;  /* Inaktiv-Zeit */
    };

   struct t_optc3 {
->     int  t_optnr;    /* Options-Nr. */
->     char *t_udatap;  /* Datenpuffer */
->     int  t_udatal;   /* Länge des Datenpuffers */
->     int  t_xdata;    /* Vorrangdaten-Auswahl */
->     int  t_timeout;  /* Inaktiv-Zeit */
->     int  t_ucepid;   /* Benutzerreferenz der Verbindung */
    };
```

t_optnr
: Optionsnummer. Anzugeben ist:

  T_OPTC1 bei *t_optc1*,
  T_OPTC3 bei *t_optc3*.

t_udatap
: Zeiger zu einem Bereich mit Benutzerdaten, die die rufende TS-Anwendung erhalten soll.

  Standardwert bei Angabe von NULL: undefiniert

t_udatal
: Länge der aus dem Bereich *t_udatap* zu übertragenden Benutzerdaten in Byte.
  Wird für *t_udatal* 0 angegeben, so wird *t_udatap* nicht ausgewertet.
  Der Maximalwert für *t_udatal* ist abhängig von der Transportverbindung.

  Standardwert bei Angabe von NULL: 0

t_xdata
: In *t_xdata* beantwortet die laufende Task den Vorschlag der rufenden TS-Anwendung zum Austausch von Vorrangdaten. Der Vorschlag wird die Task nach dem Aufruf *t_conin()* übergeben.

  Zulässige Werte sind:

  T_YES
  : Der Vorschlag der rufenden TS-Anwendung nach Vorrangdaten wird akzeptiert.

  T_NO
  : Vorrangdaten werden abgelehnt.
    Die Antwort ist verbindlich.

Hatte die rufende TS-Anwendung die Verwendung von Vorrangdaten von vornherein ausgeschlossen, muß mit T_NO geantwortet werden.

Standardwert bei Angabe von NULL: T_NO.

t_timeout
Der Inhalt dieses Feldes ist bei CMX(BS2000) stets T_NO.

t_ucepid
In diesem Feld kann eine beliebige Benutzerreferenz dieser Verbindung an CMX(BS2000) übergeben werden.
Diese Benutzerreferenz kann der laufenden Task als Option beim *t_event()* von CMX(BS2000) zurückgeliefert werden.
Damit kann die laufende Task, falls sie mehrere Verbindungen hält, Ereignisse der entsprechenden Verbindung über ein selbst gewähltes Merkmal zuordnen. Die Benutzerreferenz ist eine Alternative zu der von CMX(BS2000) gewählten Transportreferenz *tref*.

Standardwert bei Angabe von NULL: 0

Rückgabewert

T_OK
Der Aufruf war erfolgreich.
T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_conin(), t_error(), t_event()

Hinweis

Die Speicherbereiche **tref, *opt, *t_udatap* müssen allokiert und vom Programm lesend zugreifbar sein - andernfalls bricht das Programm mit Adreßfehler ab! Daß Benutzerdaten angegeben sind, erkennt CMX(BS2000) an *t_udatal* > 0,
t_udatap = NULL ist erlaubt.

t_datago
Datenfluß freigeben (data go)

t_datago() gibt die gesperrte Datenanzeige auf der angegebenen Verbindung frei. Die laufende Task teilt CMX(BS2000) dadurch mit, daß sie wieder bereit ist, Daten entgegenzunehmen. Dieser Aufruf gibt auch die Vorrangdatenanzeige (sofern vereinbart) wieder frei, falls sie (auch) gesperrt war. Der Aufruf bewirkt, daß die laufende Task wieder die Ereignisse T_DATAIN und T_XDATIN für die angegebene Verbindung zugestellt erhält, falls sie anstehen.

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

-> tref
: Zeiger zu einem Feld mit der Transportreferenz der Verbindung, auf der der Datenfluß freigegeben werden soll.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

Hinweis

In CMX(BS2000) ist die Datenflußkontrolle am Interface entkoppelt von der Datenflußkontrolle auf der Transportverbindung. Das heißt, *t_datastop()* - *t_datago()* wird von der anderen TS-Anwendung erst dann bemerkt, wenn die Flußkontrolle der Transportverbindung reagiert.

t_datain
Daten empfangen (data indication)

t_datain() übernimmt ein zuvor mit *t_event()* angezeigtes Ereignis T_DATAIN.

Die laufende Task übernimmt mit *t_datain()* auf der angegebenen Verbindung Daten einer Transport Interface Data Unit (TIDU), die zur laufenden Transport Service Data Unit (TSDU) der sendenden TS-Anwendung gehört.

Die maximale Länge der TIDU ist abhängig von der verwendeten Transportverbindung. Sie kann für eine bereits aufgebaute Verbindung mit *t_info()* abgefragt werden.
Keine TIDU muß vollständig gefüllt sein. Die Aufteilung einer TSDU in TIDUs ist rein lokal und erlaubt keine Rückschlüsse auf die Aufteilung der TSDU in TIDUs bei der sendenden TS-Anwendung.
Zwischen zwei TIDUs derselben TSDU können beliebige andere CMX(BS2000)-Ereignisse für dieselbe oder eine andere Verbindung auftreten.

Beim Aufruf *t_datain()* wird ein zusammenhängender Datenbereich *datap* bereitgestellt, in den CMX(BS2000) die Daten der empfangenen TIDU einträgt.

t_datain() zeigt an:

– (im Parameter *chain*)
ob noch eine weitere TIDU zur laufenden TSDU gehört (*chain* = T_MORE) oder nicht (*chain* = T_END).
Die einzelnen TIDUs einer TSDU werden jeweils bei *t_event()* mit dem Ereignis T_DATAIN angezeigt.

– (mit dem Ergebniswert)
ob die aktuelle TIDU vollständig gelesen wurde oder nicht.
Wird der Wert T_OK zurückgegeben, so paßte die TIDU in den bereitgestellten Datenbereich hinein. Die laufende Task hat die aktuelle TIDU vollständig übernommen.
Wird ein Wert n > 0 zurückgegeben, so wurde nur ein Teil der TIDU gelesen. n ist die Anzahl der Byte, die aus der TIDU noch nicht gelesen wurden (Restlänge). In diesem Fall muß zunächst *t_datain()* oder *t_vdatain()* solange aufgerufen werden, bis die ganze TIDU gelesen ist. Erst dann sind wieder andere CMX(BS2000)-Aufrufe möglich, z.B. *t_event()*.

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

-> tref
Zeiger zu einem Feld mit der bei *t_event()* erhaltenen Transportreferenz der Verbindung.

<- datap
Zeiger auf einen Bereich, in den CMX(BS2000) die Daten der empfangenen TIDU einträgt.

<> datal
Vor dem Aufruf ist für *datal* der Zeiger zu einem Feld anzugeben, in das die Länge von *datap* eingetragen werden muß (mindestens 1). Nach dem Aufruf liefert CMX(BS2000) in diesem Feld die Anzahl der Byte zurück, die in den Bereich *datap* eingetragen wurden. Dies muß nicht die maximale Länge der TIDU sein.

<- chain
chain ist der Zeiger zu einem Feld, in das CMX(BS2000) einen Indikator zurückliefert. Dieser Indikator zeigt an, ob noch eine weitere TIDU zur TSDU gehört.

Mögliche Werte:

T_MORE
Es folgt noch eine weitere zur TSDU gehörende TIDU. Sie wird mit einem eigenen T_DATAIN angezeigt.
T_END
Die vorliegende TIDU ist die letzte der TSDU.

Rückgabewert

T_OK
Der Aufruf war erfolgreich. Die TIDU wurde vollständig gelesen.
n > 0
Es sind noch n Byte in der TSDU vorhanden.
T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

t_datarq
Daten senden (data request)

t_datarq() sendet auf der angegebenen Verbindung die nächste (einzige) Transport Interface Data Unit (TIDU) einer Transport Service Data Unit (TSDU) an die empfangende TS-Anwendung.

Die TIDU, die von *t_datarq()* übertragen werden soll, muß von der laufenden Task in einem zusammenhängenden Datenbereich bereitgestellt werden.

Wenn die zu sendende TSDU länger ist als eine TIDU, muß sie mit mehreren Aufrufen *t_datarq()* (bzw. *t_vdatarq()*) hintereinander übermittelt werden. Deshalb muß die sendende Task bei jedem *t_datarq()* im Parameter *chain* angeben, ob noch weitere TIDUs folgen, die zur selben TSDU gehören.

Die maximale Länge der TIDU hängt ab von der verwendeten Transportverbindung. Sie kann für eine etablierte Verbindung mit *t_info()* abgefragt werden.

Liefert *t_datarq()* den Wert T_DATASTOP zurück, so ist die TIDU von CMX(BS2000) übernommen, der Fluß der TIDUs aber für diese Verbindung gesperrt worden.

Der Fluß der TIDUs kann gesperrt werden von:

– der empfangenden TS-Anwendung;
  sie kann den Fluß der TIDUs durch Aufruf von *t_datastop()* oder *t_xdatstop()* sperren,

– CMX(BS2000),
  wenn der lokale Zwischenspeicher voll ist.

Ist der Fluß der TIDUs gesperrt, so muß mit *t_event()* erst das Ereignis T_DATAGO für die Verbindung abgewartet werden, bevor die nächste TIDU gesendet werden kann.

Ein erfolgreicher Abschluß von *t_datarq()* (T_OK) bedeutet nicht, daß die empfangende TS-Anwendung die Daten bereits entgegengenommen hat.
Ein erfolgloser Abschluß von *t_datarq()* (T_ERROR) bedeutet stets, daß lokal ein Fehler erkannt wurde.

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

-> tref
Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

-> datap
Zeiger auf den Datenbereich, der die zu sendende TIDU enthält.

-> datal
Zeiger zu einem Feld mit der Anzahl der zu sendenden Byte aus dem Bereich *datap*. Anzugeben ist mindestens 1 und maximal die maximale Länge einer TIDU.

<- chain
Zeiger zu einem Indikator, durch den die Task anzeigt, ob noch eine weitere TIDU zur TSDU gehört.

Mögliche Werte:

T_MORE
Es folgt noch eine weitere TIDU, die zur TSDU gehört.
T_END
Die vorliegende TIDU ist die letzte der TSDU.

Rückgabewert

T_OK
Der Aufruf war erfolgreich, weitere TIDUs können sofort gesendet werden.
T_DATASTOP
Der Aufruf war erfolgreich, weitere TIDUs dürfen erst gesendet werden, wenn für diese Verbindung das Ereignis T_DATAGO eingetroffen ist.
T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

Hinweise

Es ist verboten, nach T_DATASTOP "auf Verdacht" weiterzusenden. CMX(BS2000) verhindert dies zwar nicht, Zuwiderhandelnde müssen aber damit rechnen, daß später zu viele oder bereits wieder ungültige T_DATAGO-Ereignisse eintreffen.

Der Speicherbereich **datap* muß allokiert und vom Programm aus lesend zugreifbar sein, sonst wird das Programm mit Adreßfehler unterbrochen. NULL ist für *datap* eine gültige Adresse.

t_datastop
Datenanzeige sperren (data stop)

t_datastop() sperrt die Datenanzeige für Normaldaten auf der angegebenen Verbindung.

t_datastop() bewirkt im einzelnen:

– Die laufende Task teilt CMX(BS2000) so mit, daß sie bis auf weiteres nicht bereit ist, für diese Verbindung Daten zu empfangen. Ein bereits angezeigtes Ereignis T_DATAIN muß aber erst beantwortet werden.

– Die laufende Task bekommt das Ereignis T_DATAIN für die angegebene Verbindung nicht mehr zugestellt. Sie kann aber, während die Datenanzeige gesperrt ist, andere CMX(BS2000)-Funktionen aufrufen, z.B. eine weitere Verbindung aufbauen, abbauen oder umlenken.

– Die sendende TS-Anwendung erhält (im Verlauf) bei *t_datarq()* das Ergebnis T_DATASTOP. Sie darf keine Daten mehr senden.

Freigegeben wird die Datenanzeige mit *t_datago()*.
Vorrangdaten sind von *t_datastop()* nicht betroffen.

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

-> tref
Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

Rückgabewert

T_OK
Der Aufruf war erfolgreich.
T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

Hinweis

Im Gegensatz zu CMX(SINIX) akzeptiert CMX(BS2000) den Aufruf *datastop()* auch dann, wenn T_DATAIN für diese Verbindung angezeigt und die Daten noch nicht komplett abgeholt wurden. Diese müssen jedoch noch mit einem *t_datain()* bzw. *v_datain()*-Aufruf abgeholt werden.

t_detach
Abmelden Task aus TS-Anwendung (detach task)

t_detach() meldet die laufende Task aus der TS-Anwendung ab, die beim Parameter *name* angegeben ist. Falls noch Verbindungen dieser Task existieren, werden sie implizit abgebaut. Im Normalfall sollten jedoch alle Verbindungen der Task vor dem Aufruf von *t_detach()* mit *t_disrq()* abgebaut werden.

Meldet sich die letzte Task einer TS-Anwendung ab, so wird die TS-Anwendung aufgelöst. Verbindungswünsche für diese TS-Anwendung werden dann nicht mehr angenommen. Existieren noch weitere Tasks für diese TS-Anwendung, so erhält ab diesem Zeitpunkt die Task, die sich nach der aktuellen Task an die TS-Anwendung angemeldet hat, die Verbindungswünsche zugestellt.

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

-> name
: Zeiger zu einer Struktur *t_myname* mit dem LOKALEN NAMEN der TS-Anwendung. Es ist derselbe LOKALE NAME anzugeben wie bei *t_attach()*.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_attach(), t_error()

t_disin
Verbindungsabbau entgegennehmen (disconnection indication)

t_disin() nimmt ein zuvor bei *t_event()* gemeldetes Ereignis T_DISIN entgegen. T_DISIN zeigt an, daß die Verbindung abgebaut wurde.

t_disin() gibt an, ob die ferne TS-Anwendung oder das Transportsystem das Ereignis T_DISIN ausgelöst hat.

t_disin() liefert ferner:

– die Benutzerdaten, die die ferne TS-Anwendung mitgeschickt hat, falls das Ereignis T_DISIN von der fernen TS-Anwendung ausgelöst wurde und sofern die Transportverbindung diese Option bietet.

– den Grund für den Abbau der Transportverbindung, falls das Ereignis T_DISIN von CMX(BS2000) oder vom Transportsystem ausgelöst wurde.
Der Grund für den Verbindungsabbau wird von *t_disin()* in sedezimaler Darstellung geliefert. Die Klartextdarstellung des Codes erhält man mit Hilfe von *t_preason()* bzw. *t_strreason()*.

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

-> tref
Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

<- reason
Zeiger zu einem Feld, in das CMX(BS2000) den Grund des Verbindungabbaus einträgt.

Mögliche Werte:

T_USER
Die Verbindung wurde durch die ferne TS-Anwendung abgebaut.

sonst:
Die Verbindung wurde durch CMX(BS2000) oder das Transportsystem abgebaut. Die Verbindungsabbaugründe sind weiter unten kommentiert.

<> opt

Für *opt* ist der Wert NULL oder der Zeiger auf eine Variante anzugeben, die eine Struktur mit Systemoptionen enthält.
Damit können die Benutzerdaten abgefragt werden, die die ferne TS-Anwendung beim Verbindungsabbau angegeben hat.
Wird *opt* = NULL angegeben, so vernichtet CMX(BS2000) die Benutzerdaten. Hat die ferne TS-Anwendung keine Benutzerdaten angegeben, so liefert CMX(BS2000) die angegebenen Standardwerte. Die Übermittlung der Benutzerdaten wird beim Verbindungsabbau nicht garantiert, sie hängt vom der zugrundeliegenden Transportverbindung ab.

Folgende Struktur ist in <*cmx.h*> definiert:

```
    struct t_optc2 {
->      int  t_optnr;    /* Options-Nr. */
<-      char *t_udatap;  /* Datenpuffer */
< >     int  t_udatal;   /* Länge des Datenpuffers */
     };
```

t_optnr
: Optionsnummer. Anzugeben ist T_OPTC2.

t_udatap
: Zeiger zu einem Datenbereich, in den CMX(BS2000) die empfangenen Benutzerdaten der fernen TS-Anwendung überträgt. Der Bereich muß dann so groß sein, daß die empfangenen Daten ganz hineinpassen. Die maixmal zulässige Länge der Benutzerdaten hängt von der verwendeten Transportverbindung ab. T_DIS_SIZE ist eine für alle Transportverbindungen geeignete Maximalgröße.

  Standardwert bei Angabe von NULL: undefiniert

t_udatal
: Vor dem Aufruf muß hier 0 oder die Länge des Datenbereiches *t_udatap* stehen. Nach dem Aufruf liefert CMX(BS2000) in diesem Feld die Anzahl der nach *t_udatap* übertragenen Byte zurück.

  Standardwert bei Angabe von NULL: 0.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

Abbaugründe

Die in *reason* gelieferten Verbindungsabbaugründe haben nachstehende Bedeutung. Die angegebenen symbolischen Werte sind in *cmx.h* definiert; im Zweifelsfall gilt der in *cmx.h* definierte numerische Wert

```
T_USER         0 Auf Anforderung des Partners wurde die Verbindung abgebaut.
T_RUNKNOWN   256 Abbau vom fernen Transportsystem, Grund nicht angegeben.
T_RPERMLOST  261 Abbau durch Administration des lokalen Transportsystems.
T_RSYSERR    262 Abbau vom Transportsystem aufgrund eines Neztwerkfehlers.
T_RCONGEST   385 Abbau vom fernen Transportsystem aufgrund von Betriebs-
                 mittelengpaß.
T_RNOCONN    392 Aufbau der Netzverbindung vom fernen Transportsystem
                 abgelehnt.
T_RLCONGEST  448 Abbau vom lokalen Transportsystem wegen Betriebsmittelengpaß.
T_RLPROTERR  464 Abbau vom lokalen Transportsystem wegen Transportprotokoll-
                 fehler (Systemfehler).
T_RLPERMLOST 481 Abbau durch Administration des lokalen Transportsystems.
```

Hinweise

Die Speicherbereiche müssen allokiert und vom Programm lesend (**tref*) bzw. schreibend (**reason, *opt, *t_udatap*) zugreifbar sein, andernfalls bricht das Programm mit Adreßfehler ab! Daß Benutzerdaten angegeben sind, erkennt CMX(BS2000) an *t_udatal* > 0, *t_udatap* = NULL ist erlaubt!.

Bei Verbindungsabbau ist die *tref* bereits zum Zeitpunkt des Verbindungsabbaus für alle Aufrufe ungültig. Nur für *t_event(tref) und t_disin(tref)* ist sie noch gültig.

t_disrq
Verbindung abbauen (disconnection request)

t_disrq() baut die angegebene Verbindung ab oder weist die Verbindungsaufbauanzeige einer rufenden TS-Anwendung ab. In beiden Fällen erhält die ferne TS-Anwendung eine Verbindungsabbauanzeige mit dem Grund T_USER.

Jeder Partner kann die Verbindung abbauen, unabhängig davon, welcher sie aktiv aufgebaut hat.

Mit dem Verbindungsabbau können der fernen TS-Anwendung Benutzerdaten mitgeschickt werden, falls die Transportverbindung diese Option bietet.

t_disrq() kann Daten, die noch unterwegs sind, überholen. Diese gehen dann verloren.

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

-> tref
: Zeiger zu einem Feld mit der Transportreferenz der Verbindung, die abgebaut werden soll.

-> opt
: Für *opt* ist der Wert NULL oder der Zeiger auf eine Variante anzugeben, die eine Struktur mit Systemoptionen enthält.
Damit werden die Benutzerdaten angegeben, die die ferne TS-Anwendung mit der Anzeige des Verbindungsabbaus erhalten soll.
Wird *opt* = NULL angegeben, so nimmt CMX(BS2000) die angegebenen Standardwerte.

Folgende Struktur ist in <*cmx.h*> definiert:

```
    struct t_optc2 {
->      int  t_optnr;    /* Options-Nr. */
->      char *t_udatap;  /* Datenpuffer */
->      int  t_udatal;   /* Länge des Datenpuffers */
    };
```

t_optnr
: Optionsnummer. Anzugeben ist T_OPTC2.

t_udatap
: Zeiger zu einem Bereich mit Benutzerdaten, die die ferne TS-Anwendung erhalten soll.

  Standardwert bei Angabe von NULL: undefiniert

t_udatal
: Länge der aus dem Bereich t_udatap zu übertragenden Benutzerdaten.
  Wird *t_udatal* = 0 angegeben, so wird *t_udatap* nicht ausgewertet.
  Der Maximalwert für *t_udatal* ist abhängig von der Transportverbindung.

  Standardwert bei Angabe von NULL: 0

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

t_error
Fehlerdiagnose (error)

t_error() liefert **Diagnoseinformationen**, wenn ein anderer CMX(BS2000)-Aufruf das Ergebnis T_ERROR bzw. NULL hatte.

Die möglichen Fehlermeldungen zu den Aufrufen an der Programmschnittstelle ICMX entstehen entweder in den Funktionen der CMX(BS2000)-Bibliothek im Benutzerprozeß oder im Betriebssystem. Sie müssen danach unterschieden werden, ob sie in CMX(BS2000) selbst erzeugt werden oder aus Betriebssystemaufrufen in CMX(BS2000) resultieren.
Die in CMX(BS2000) entstandenen Fehlermeldungen werden in sedezimaler Darstellung von *t_error()* übergeben. Dieser Fehlercode kann mit Hilfe der Aufrufe *t_strerror()* bzw. *t_perror()* in Klartext übersetzt werden. *t_strerror()* liefert einen Zeiger auf einen statischen Bereich, der die Klartextdarstellung der Fehlermeldung erhält.
t_perror() schreibt den Klartext der Fehlermeldung nach *stderr*.

Die Fehlermeldungen von CMX(BS2000) sind im Anhang beschrieben.

#include <cmx.h>
int t_error()

Rückmeldungen

Siehe Anhang ab Seite 149.

Dateien

<cmx.h> – globale CMX(BS2000)-Definitionsdatei

Siehe auch

t_perror(), t_strerror()

t_event
Ereignis abwarten oder abfragen (event)

t_event() stellt fest, ob ein CMX(BS2000)-Ereignis für die laufende Task eingetroffen ist.

Über den Parameter *cmode* kann man den Verarbeitungsmodus von *t_event()* festlegen. *t_event()* kann:

– **synchron** darauf warten, daß ein CMX(BS2000)-Ereignis für die laufende Task eintrifft. Die Task wird während dieser Wartezeit suspendiert.
Der Wartezustand wird im BS2000 nicht durch Signale unterbrochen. Dafür muß in der Signalroutine *t_wake()* aufgerufen werden. In den Optionen *opt* kann man eine Zeitschranke für das synchrone Warten angeben. Ist innerhalb dieser Wartezeit kein Ereignis eingetroffen, wird der Wartezustand abgebrochen.

– **asynchron** prüfen, ob ein CMX(BS2000)-Ereignis für die laufende Task vorliegt. Die Funktion kehrt immer sofort zur laufenden Task zurück.

Neben dem entsprechenden Ereignis liefert *t_event()*:

– die Transportreferenz der betroffenen Verbindung für die Zuordnung Ereignis - Verbindung (Parameter *tref*),

– ereignisspezifische Zusatzinformationen, falls die Option *opt* angegeben wurde.

t_event() erlaubt CMX(BS2000) außerdem, weitere Daten für eine Verbindung anzuzeigen, wenn die Datenanzeige nicht explizit für diese Verbindung durch *t_datastop()* bzw. *t_xdatstop()* gesperrt wurde.
Ist für eine Task ein Ereignis T_DATAIN oder T_XDATIN angezeigt, so darf die betroffene Verbindung nicht umgelenkt werden. Insbesondere darf *t_event()* erst wieder aufgerufen werden, wenn die laufende Task die damit angezeigten Daten mit *t_datain()*, *t_vdatain()* bzw. *t_xdatin()* entgegengenommen hat.

Liegen mehrere Ereignisse für eine Verbindung vor, so werden sie nacheinander in der Reihenfolge angezeigt, in der sie aufgetreten sind.

Ausnahmen:

– Das Ereignis T_XDATIN (Vorrangdaten-Empfangs-Anzeige) kann Ereignisse T_DATAIN (Normaldaten-Empfangs-Anzeige) überholen, ohne sie zu zerstören.

– Das Ereignis T_DISIN (Verbindungsabbau-Anzeige) kann die Ereignisse T_DATAIN und T_XDATIN für die betroffene Verbindung überholen und damit zerstören.
Die Daten, die T_DATAIN bzw. T_XDATIN anzeigen sollte, gehen verloren.

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

<- tref

Zeiger zu einem Feld, in dem CMX(BS2000) die verbindungsspezifische Transportreferenz zurückliefert. Sie gibt an, zu welcher Verbindung das Ereignis gehört. Bei den Ereignissen T_NOEVENT, T_ERROR, ist der Inhalt von *tref* nicht definiert.

-> cmode

Mit Hilfe von *cmode* gibt man an, ob *t_event()* synchron auf ein Ereignis warten soll oder asynchron prüfen soll, ob ein Ereignis vorliegt.

Mögliche Werte:

T_WAIT (synchrone Verarbeitung)

Die laufende Task wird suspendiert, bis ein TS-Ereignis eintritt, die vereinbarte Wartezeit abläuft (Parameter *t_timeout* in *opt*) oder *t_wake()* für diese Task aufgerufen wird. In den letzten beiden Fällen wird dann das Ereignis T_NOEVENT geliefert. Auch mit *t_wake()* aus einer Signalroutine (Contingency) oder einer anderen Task mit der selben USER-ID läßt sich der Aufruf abbrechen.

T_CHECK (asynchrone Verarbeitung)

Die laufende Task prüft, ob ein TS-Ereignis vorliegt.

Liegt ein TS-Ereignis für die laufende Task vor, so wird ihr dieses Ereignis geliefert. Liegt kein Ereignis vor, so wird der Task das Ereignis T_NOEVENT geliefert.

-> opt

Für *opt* ist der Wert NULL oder der Zeiger auf eine Variante *t_opte1* anzugeben, die eine Struktur mit Systemoptionen enthält.
CMX(BS2000) setzt bei Angabe NULL die angegebenen Standardwerte.
Folgende Struktur ist in <*cmx.h*> definiert:

```
    struct t_opte1 {
->      int  t_optnr;    /* Options-Nr. */
<-      int  t_attid;    /* CMX-Referenz der Anmeldung */
<-      int  t_uattid;   /* Benutzerreferenz der Anmeldung */
<-      int  t_ucepid;   /* Benutzerreferenz der Verbindung */
->      int  t_timeout;  /* Zeitschranke für T_WAIT */
<-      int  t_evdat;    /* ereignisspezifische Informationen */
    };
```

t_optnr

Optionsnummer. Anzugeben ist T_OPTE1.

t_attid
: In *t_attid* liefert *t_event()* die CMX(BS2000)-interne Referenz der betroffenen Anmeldung.
Die CMX(BS2000)-Referenz wurde als Option von CMX(BS2000) auch beim *t_attach()* geliefert. Sie dient nur Verfolger- und Diagnosezwecken und wird ausschließlich zur Protokollierung verwendet.

t_uattid
: In *t_uattid* liefert *t_event()* die Benutzerreferenz der betroffenen Anmeldung.
Die Benutzerreferenz wurde als Option bei *t_attach()* an CMX(BS2000) übergeben. Eine Task, die mehrere TS-Anwendungen steuert, kann damit die TS-Ereignisse der entsprechenden Anmeldung einer TS-Anwendung zuordnen.

t_ucepid
: In *t_ucepid* liefert *t_event()* die Benutzerreferenz der betroffenen Verbindung bei den TS-Ereignissen T_CONCF, T_DATAIN, T_XDATIN, T_DATAGO, T_XDATGO und T_DISIN.
Die Benutzerreferenz wurde bei *t_conrq()*, *t_conrs()* oder *t_redin()* an CMX(BS2000) übergeben. Eine Task, die mehrere Verbindungen hält, kann damit die TS-Ereignisse der entsprechenden Verbindung zuordnen. Dieses vom Benutzer gewählte Merkmal ist eine Alternative zur Transportreferenz *tref*, die von CMX(BS2000) festgelegt wird.

t_timeout
: Bei *cmode* = T_WAIT:
Für *t_timeout* kann eine Wartezeit angegeben werden, während der *t_event()* synchron auf ein Ereignis warten soll.
Bei *cmode* = T_CHECK:
Eine Angabe für *t_timeout* wird ignoriert.

  Mögliche Angaben für *t_timeout*:

  T_NOLIMIT
  : Es wird keine Wartezeit vorgegeben. Die Task wartet (unbegrenzt) so lange, bis ein Ereignis eintritt oder *t_event()* durch *t_wake()* abgebrochen wird.

  T_NO
  : Die Task wartet nicht. Sie wird sofort mit dem vorhandenen TS-Ereignis oder mit T_NOEVENT fortgesetzt (entspricht *cmode* = T_CHECK).

  n > 0
  : Die Task wartet n Sekunden auf das Eintreffen eines CMX-Ereignisses. Die Genauigkeit beträgt dabei + - 60 sec. Tritt in diesem Zeitraum kein CMX-Ereignis für die wartende Task ein, so wird er mit dem Ereignis T_NOEVENT fortgesetzt.

  Standardwert bei Angabe von NULL: T_NOLIMIT

t_evdat
Hier liefert CMX(BS2000) ereignisspezifische Zusatzinformationen zurück.

Mögliche Information:
Bei den Ereignissen T_DATAIN und T_XDATIN wird hier die Länge der angezeigten Daten angegeben.
Bei den anderen TS-Ereignissen inclusive T_NOEVENT ist die Zusatzinformation undefiniert.

Rückgabewert

T_CONIN
Das Ereignis zeigt an, daß eine rufende TS-Anwendung eine Verbindung zur laufenden Task aufbauen will. Diese Verbindungsaufbauanzeige muß zunächst mit *t_conin()* abgeholt werden und dann mit *t_conrs()* bestätigt oder mit *t_disrq()* abgelehnt werden.

T_CONCF
Dieses Ereignis zeigt an, daß die gerufene TS-Anwendung, einen Verbindungsaufbauwunsch der laufenden Task positiv beantwortet hat.
Diese Verbindungsaufbaubestätigung muß mit *t_concf()* abgeholt werden.

T_DATAIN
Das Ereignis zeigt an, daß Daten auf der Verbindung empfangen wurden, die in *tref* angegeben ist. Die Daten müssen mit *t_datain()* oder *t_vdatain()* abgeholt werden.
CMX(BS2000) zeigt dieses Ereignis für eine Verbindung nicht an, solange auf ihr der Datenfluß gesperrt ist, d.h. wenn die empfangende Task für diese Verbindung *t_datastop()* gegeben hat.

T_DATAGO
Die lokale TS-Anwendung kann auf der in *tref* angegebenen Verbindung weiter Daten senden.
Mögliche Reaktion: *t_datarq()* oder *t_vdatarq()*.
Das Ereignis T_DATAGO erlaubt es der lokalen TS-Anwendung auch, auf dieser Verbindung wieder Vorrangdaten zu senden, sofern beim Verbindungsaufbau das Senden und Empfangen von Vorrangdaten vereinbart wurde.

T_DISIN
Dieses Ereignis zeigt den Abbau der Verbindung an, die in *tref* angegeben ist.
Diese Verbindungsabbauanzeige muß mit *t_disin()* abgeholt werden.

T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

T_NOEVENT
Das Ereignis bedeutet:

bei cmode = T_CHECK
Es liegt kein Ereignis vor.

bei cmode = T_WAIT
Wartezustand der Task abgebrochen. Der Abbruch erfolgte durch *t_wake()* oder wegen Ablauf der vereinbarten Wartezeit. Ein TS-Ereignis trat nicht auf.

Der Inhalt von *tref* ist nicht definiert.

T_REDIN
Dieses Ereignis zeigt an, daß eine andere Task derselben TS-Anwendung eine Verbindung auf die laufende Task umgelenkt hat.
Die Verbindungsumlenkung muß mit *t_redin()* abgeholt werden.

T_XDATIN
Das Ereignis zeigt an, daß Vorrangdaten auf der Verbindung empfangen wurden, die in *tref* angegeben ist. Die Daten müssen mit *t_xdatin()* abgeholt werden.

Dieses Ereignis wird nur angezeigt:

– wenn beim Verbindungsaufbau der Austausch von Vorrangdaten vereinbart wurde.

– solange auf der Verbindung der Vorrangdatenfluß nicht gesperrt ist. Der Vorrangdatenfluß ist gesperrt, wenn die empfangende Task für diese Verbindung *t_xdatstop()* gegeben hat.

T_XDATGO
CMX(BS2000) zeigt mit diesem Ereignis an, daß die Task auf der in *tref* angegebenen Verbindung weiter Vorrangdaten senden darf.
Mögliche Reaktion: *t_xdatrq()*.

CMX(BS2000) zeigt dieses Ereignis nur an, wenn beim Verbindungsaufbau der Austausch von Vorrangdaten vereinbart wurde.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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()

Hinweise

Ein angezeigtes Ereignis muß mit dem entsprechenden Aufruf sofort abgeholt werden.

Wird ein bei *t_event()* angezeigtes Ereignis nicht abgeholt, dann zeigt es CMX(BS2000) beim nächsten *t_event()* wieder an. Wurde mit *t_datain(), t_vdatain()* bzw. *t_xdatin()* nur ein Teil der angezeigten Nachricht abgeholt, dann bringt ein darauffolgender *t_event()*-Aufruf T_ERROR (T_WSEQUENCE).

CMX(BS2000) prüft bei *t_datarq(), t_vdatarq()* und *t_xdatrq()* nicht, ob ein bereits angezeigtes, aber noch nicht abgeholtes Ereignis für diese *tref* vorliegt. Z.B. muß der Anwender bei *t_event(T_REDIN)* dafür sorgen, daß vor einem *t_datarq()* der *t_redin()* aufgerufen wird.

Tritt beim Senden der Returncode T_DATASTOP bzw. T_XDATSTOP auf, dann darf erst wieder gesendet werden, nachdem für diese Verbindung beim *t_event()* das Ereignis T_DATAGO bzw. T_XDATGO angezeigt wurde.

t_getaddr
TRANSPORTADRESSE abfragen (get address)

t_getaddr() ermittelt die TRANSPORTADRESSE einer fernen TS-Anwendung.

Für den Parameter *globname* ist der GLOBALE NAME der TS-Anwendung anzugeben. *t_getaddr()* liefert als Ergebnis den Zeiger auf einen statischen Bereich mit der TRANSPORTADRESSE dieser TS-Anwendung zurück.

Dieser statische Bereich wird bei jedem Aufruf überschrieben. Wenn der Bereich gesichert werden soll, muß der Aufrufer den Bereich kopieren.
Die Länge des zu kopierenden Bereichs ergibt sich aus dem in *struct t_partaddr* definierten Längenfeld *t_palng*.

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

-> globname

Für diesen Parameter ist der GLOBALE NAME der TS-Anwendung anzugeben, deren TRANSPORTADRESSE ermittelt werden soll.
Der GLOBALE NAME ist als mit NIL endender String in der Form

"NP5.NP4.NP3.NP2.NP1"

anzugeben.
Die NPi (i=1,2,3,4,5) sind die Namensteile des hierarchischen GLOBALEN NAMENS. Dabei ist NP5 der Namensteil[5], also der Namensteil der unteren Hierarchiestufe. NP1 ist der Namensteil[1], also der in der Hierarchie höchste Namensteil. Die restlichen Namensteile sind in von links nach rechts aufsteigender hierarchischer Reihenfolge anzugeben.

Ist bei einem GLOBALEN NAMEN einer der Namensteile nicht belegt (z.B. NP4) und folgt diesem Namensteil noch ein Namensteil höherer Hierarchie (z.B. NP3), so ist von dem nicht belegten Namensteil das Trennzeichen (.) anzugeben.
Eine Folge von Trennzeichen am Ende des Wertes von *globname* kann weggelassen werden.
Der GLOBALE NAME wird dann wie folgt angegeben: "NP5..NP3"

Mindestens einer der Namensteile NPi muß angegeben werden.

Ist das Trennzeichen Punkt (.) Bestandteil eines Namensteils, so muß es als \. (Gegenschrägstrich Punkt) dargestellt werden.

Beispiel:

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

   Angabe für *globname*:
   "G\.MEIER.DF1.MCH-P.SIEMENS-AG.D"

2. GLOBALE NAME:
   Namensteil[2] = BU&B
   Namensteil[5] = PENCILPUSHER

   Angabe für *globname*:
   "PENCILPUSHER...BU&B"

-> opt
Der Wert von *opt* muß NULL sein.

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_getaddr()* den Zeiger auf einen Bereich mit der TRANSPORTADRESSE zurück.

Im Falle eines Fehlers liefert *t_getaddr()* den NULL-Zeiger zurück. Der Fehlercode kann mit *t_error()* abgefragt werden.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_error()

t_getloc
LOKALEN NAMEN abfragen (get local name)

t_getloc() ermittelt den LOKALEN NAMEN einer TS-Anwendung.

Für den Parameter *globname* ist der GLOBALE NAME der TS-Anwendung anzugeben. *t_getloc()* liefert als Ergebnis den Zeiger auf einen statischen Bereich mit dem LOKALEN NAMEN dieser TS-Anwendung zurück.

Dieser statische Bereich wird bei jedem Aufruf überschrieben. Wenn der Bereich gesichert werden soll, muß der Aufrufer den Bereich kopieren.
Die Länge des zu kopierenden Bereichs ergibt sich aus dem in *struct t_myname* definierten Längenfeld *t_mnlng*.

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

-> globname

Für diesen Parameter ist der GLOBALE NAME der TS-Anwendung anzugeben, deren LOKALER NAMEN ermittelt werden soll.
Der GLOBALE NAME ist als mit NIL endender String in der Form

"NP5.NP4.NP3.NP2.NP1"

anzugeben.
Die NPi (i=1,2,3,4,5) sind die Namensteile des GLOBALEN NAMENS. Dabei ist NP5 der Namensteil[5], also der Namensteil der unteren Hierarchiestufe. NP1 ist der Namensteil[1], also der in der Hierarchie höchste Namensteil. Die restlichen Namensteile sind in von links nach rechts aufsteigender hierarchischer Reihenfolge anzugeben.

Ist bei einem GLOBALEN NAMEN einer der Namensteile nicht belegt (z.B. NP4) und folgt diesem Namensteil noch ein Namensteil höherer Hierarchie (z.B. NP3), so ist von dem nicht belegten Namensteil das Trennzeichen (.) anzugeben. Eine Folge von Trennzeichen am Ende des Wertes von *globname* kann weggelassen werden.
Der GLOBALE NAME wird dann wie folgt angegeben: "NP5..NP3"

Mindestens einer der Namensteile NPi muß angegeben werden.

Ist das Trennzeichen . (Punkt) Bestandteil eines Namensteils, so muß es als \. (Gegenschrägstrich Punkt) dargestellt werden.

Beispiel:

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

   Angabe für *globname*:
   "G\.MEIER.DF1.MCH-P.SIEMENS-AG.D"

2. GLOBALER NAME:
   Namensteil[2] = BU&B
   Namensteil[5] = PENCILPUSHER

   Angabe für *globname*:
   "PENCILPUSHER...BU&B"

-> opt
Der Wert von *opt* muß NULL sein.

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_getloc()* den Zeiger auf einen Bereich mit dem LOKALEN NAMEN zurück.
Im Falle eines Fehlers liefert *t_getloc()* den NULL-Zeiger zurück. Der Fehlercode kann mit *t_error()* abgefragt werden.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_error()

t_getname
GLOBALEN NAMEN abfragen (get name)

t_getname() ermittelt zur TRANSPORTADRESSE einer fernen TS-Anwendung deren GLOBALEN NAMEN aus dem TS-Directory 1.

Die TRANSPORTADRESSE der TS-Anwendung muß vom Aufrufer im Parameter *addr* angegeben werden.

t_getname() liefert als Ergebnis den Zeiger auf einen statischen Bereich mit dem GLOBALEN NAMEN dieser TS-Anwendung zurück.

Dieser statische Bereich wird bei jedem Aufruf überschrieben. Wenn der Bereich gesichert werden soll, muß der Aufrufer den Bereich kopieren.

Der GLOBALE NAME wird von CMX(BS2000) als mit NIL endender String in der Form

NP5.NP4.NP3.NP2.NP1

geliefert. Die NPi (i=1,2,3,4,5) sind die Namensteile des GLOBALEN NAMENS. Dabei ist NP5 der Namensteil[5], also der Namensteil der unteren Hierarchiestufe. NP1 ist der Namensteil[1], also der in der Hierarchie höchste Namensteil. Die restlichen Namensteile sind in von links nach rechts aufsteigender hierarchischer Reihenfolge angegeben.

Ist bei einem GLOBALEN NAMEN einer der Namensteile nicht belegt (z.B. NP4) und folgt diesem Namensteil noch ein Namensteil höherer Hierarchie (z.B. NP3), so wird von dem nicht belegten Namensteil das Trennzeichen (.) dennoch geliefert.
Eine Folge von Trennzeichen am Ende des Wertes von *globname* wird weggelassen.
Der GLOBALE NAME wird dann von CMX(BS2000) wie folgt angegeben: "NP5..NP3"

Ist das Trennzeichen . (Punkt) Bestandteil eines Namensteils, so wird es als \. (Gegenschrägstrich Punkt) dargestellt.

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

-> addr
    Zeiger auf einen Bereich mit der TRANSPORTADDRESSE

-> opt
    Der Wert von *opt* muß NULL sein.

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_getname()* den Zeiger auf einen Bereich mit dem GLOBALEN NAMEN zurück.
Im Falle eines Fehlers liefert *t_getname()* den NULL-Zeiger zurück. Der Fehlercode kann mit *t_error()* abgefragt werden.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_error()

t_info
Informationen über CMX(BS2000) abfragen (information)

t_info() informiert CMX(BS2000) über die Länge der Transport Interface Data Unit (TIDU) für die angegebene Verbindung.

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

-> tref
Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

<- opt
Zeiger zu einer Variante mit Benutzeroptionen. Folgende Struktur ist in <*cmx.h*> definiert:

```
   struct t_opti1 {
->      int t_optnr;    /* Options-Nr. */
<-      int t_maxl;     /* TIDU-Länge */
   };
```

t_optnr
Optionsnummer. Anzugeben ist T_OPTI1.

t_maxl
In dieses Feld trägt CMX(BS2000) die Länge der TIDU ein.
Der Wert gibt an, wieviele Byte beim Datentransfer über diese Verbindung maximal pro Aufruf an CMX(BS2000) übergeben bzw. von CMX(BS2000) empfangen werden können.

Rückgabewert

T_OK
Der Aufruf war erfolgreich.

T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

t_perror
CMX(BS2000)-Fehlermeldung decodiert ausgeben

t_perror() decodiert CMX(BS2000)-Fehlermeldungen, die der Task beim Aufruf von *t_error()* in sedezimaler Darstellung von CMX(BS2000) übergeben wurden.
t_perror() schreibt den Klartext zu der in *code* angegebenen CMX(BS2000)-Fehlermeldung auf die Standardfehlerausgabe *stderr*.

Im Parameter *s* kann ein zusätzlicher erklärender Text angegeben werden, z.B Angabe auf welchen CMX(BS2000)-Aufruf und welche TS-Anwendung sich der Fehler bezieht.

Aufbau der Ausgabe von *t_perror()*:

t_perror() schreibt zunächst den durch *s* angegebenen Text (sofern s != NULL), dann : (Doppelpunkt) und \n (Zeilenende). Danach folgt der Klartext der übergebenen CMX(BS2000)-Fehlermeldung. Der Klartext setzt sich zusammen aus den Fehlersymbolen, wie in <*cmx.h*> definiert. Jedes Fehlersymbol wird von \t eingeleitet.

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

-> s
Zeiger auf einen Bereich mit dem Text, der der Klartextdarstellung der Fehlermeldung vorangestellt werden soll, oder der Wert NULL.

-> code
Für *code* ist die Darstellung der Fehlermeldung anzugeben, die der Task beim Aufruf von *t_error()* von CMX(BS2000) übergeben wurde.

Siehe auch

t_error(), t_strerror()

t_preason
Verbindungsabbaugründe decodieren und ausgeben

t_preason() decodiert Verbindungsabbaugründe, die der Task beim Aufruf von *t_disin()* in sedezimaler Darstellung übergeben werden.
t_preason() schreibt die Klartextdarstellung zu dem in *reason* übergebenen Verbindungsabbaugrund auf die Standardfehlerausgabe *stderr*.

Im Parameter *s* kann ein zusätzlicher erklärender Text angegeben werden, z.B Angabe auf welche Verbindung oder auf welche TS-Anwendung sich die Ausgabe bezieht.

Aufbau der Ausgabe von *t_preason()*:

t_reason() schreibt zunächst den durch *s* angegebenen Text (sofern s != NULL), dann : (Doppelpunkt) und \n (Zeilenende). Danach folgt der Klartext zu dem übergebenen Verbindungsabbaugrund. Der Klartext setzt sich zusammen aus dem Symbol für den Verbindungsabbaugrund, wie in <*cmx.h*> definiert. Das Symbol für den Verbindungsabbau wird von \t eingeleitet.

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

-> s
Zeiger auf einen Bereich mit dem Text, der der Klartextdarstellung des Verbindungsabbaugrundes vorangestellt werden soll, oder der Wert NULL.

-> reason
Für *reason* ist die Darstellung des Verbindungsabbaugrundes anzugeben, die der Task beim Aufruf von *t_disin()* von CMX(BS2000) übergeben wurde.

Siehe auch

t_disin(), t_strreason()

t_redin
Umgelenkte Verbindung annehmen (redirection indication)

t_redin() nimmt ein zuvor mit *t_event()* gemeldetes Ereignis T_REDIN entgegen. T_REDIN zeigt an, daß eine andere Task derselben TS-Anwendung eine Verbindung auf die laufende Task umgelenkt hat.

Das Ereignis T_REDIN **muß** mit *t_redin()* entgegengenommen werden. Ist die Verbindung unerwünscht, kann sie mit *t_redrq()* der ursprünglichen Task zurückgegeben oder mit *t_disrq()* abgebaut werden.

Der Aufruf *t_redin()* liefert:

– die TSN der rufenden Task,

– die Benutzerdaten, die die rufende Task bei der Umlenkung mitgeschickt hat.

Ist die laufende Task in mehreren TS-Anwendungen angemeldet, so muß sie selbst durch geeignete Mittel feststellen, zu welcher TS-Anwendung die umgelenkte Verbindung gehört. Geeignete Mittel sind z.B. die Benutzerdaten oder die bei *t_event()* gelieferte optionale Benutzerreferenz der Anmeldung dieser TS-Anwendung.

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

-> tref
Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

<- pid
Zeiger zu einem Feld, in dem CMX(BS2000) die Taskidentifikation der Task zurückliefert, die die Verbindung umlenkt.

<> opt
Für *opt* ist der NULL-Zeiger oder der Zeiger zu einer Variante mit Systemoptionen anzugeben.
Damit können Benutzerdaten entgegengenommen werden, die die rufende Task bei der Anforderung der Verbindungsumlenkung (*t_redrq()*) mitgegeben hat.
Wird *opt* = NULL angegeben, so vernichtet CMX(BS2000) die Benutzerdaten.

Hat die rufende Task keine Benutzerdaten angegeben, so liefert CMX(BS2000) die angegebenen Standardwerte.

Folgende Struktur ist in <*cmx.h*> definiert:

```
    struct t_optc2 {
->      int  t_optnr;     /* Options-Nr. */
<-      char *t_udatap;   /* Datenpuffer */
< >     int  t_udatal;    /* Länge des Datenpuffers */
    };
   struct t_optc3 {
->     int  t_optnr;    /* Options-Nr. */
<-     char *t_udatap;  /* Datenpuffer */
< >    int  t_udatal;   /* Länge des Datenpuffers */
<-     int  t_xdata;    /* Vorrangdaten-Auswahl */
<-     int  t_timeout;  /* Inaktiv-Zeit */
->     int  t_ucepid;   /* Benutzerreferenz der Verbindung */
    };
```

t_optnr
: Optionsnummer. Anzugeben ist:

  T_OPTC2 bei *t_optc2*
  T_OPTC3 bei *t_optc3*

t_udatap
: Zeiger auf einen Datenbereich, in den CMX(BS2000) die empfangenen Benutzerdaten einträgt.

  Standardwert bei Angabe von NULL: undefiniert

t_udatal
: Vor dem Aufruf muß hier 0 oder die Länge des Datenbereiches *t_udatap* stehen. Sie muß so groß sein, daß die empfangenen Daten ganz hineinpassen. Als geeignete Maximalgröße ist T_RED_SIZE in <*cmx.h*> definiert. CMX(BS2000) liefert in diesem Feld die Anzahl der empfangenen Byte zurück.

  Standardwert bei Angabe von NULL: 0.

t_xdata
: In *t_xdata* wird immer der Wert T_NO zurückgeliefert.

t_timeout
: In *t_timeout* wird immer der Wert T_NO zurückgeliefert.

t_ucepid
: In diesem Feld kann eine beliebige Benutzerreferenz dieser Verbindung an CMX(BS2000) übergeben werden.
  Diese Benutzerreferenz kann der laufenden Task als Option im folgenden beim *t_event()* von CMX(BS2000) zurückgeliefert werden.

Damit kann die laufende Task, falls sie mehrere Verbindungen hält, Ereignisse der entsprechenden Verbindung über ein selbst gewähltes Merkmal zuordnen. Die Benutzerreferenz ist eine Alternative zu der von CMX(BS2000) gewählten Transportreferenz *tref*.

Standardwert bei Angabe von NULL: 0

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

Hinweis

Die Speicherbereiche müssen allokiert und vom Programm lesend *(*tref)* bzw. schreibend (**pid, *opt, *t_udatap*) zugreifbar sein, andernfalls bricht das Programm mit Adreßfehler ab! Daß Benutzerdaten angegeben sind, erkennt CMX(BS2000) an *t_udatal* > 0, *t_udatap* = NULL ist erlaubt!

t_redrq
Verbindung umlenken (redirection request)

t_redrq() lenkt die angegebene Verbindung an eine andere Task um. Die empfangende Task wird durch die TSN festgelegt. Sie muß zum Zeitpunkt der Umlenkung in der TS-Anwendung angemeldet sein, zu der die umzulenkende Verbindung gehört.

Beim *t_redrq()* kann die laufende Task in der Option *opt* Benutzerdaten angeben, die der empfangenden Task bei der Übernahme der Verbindung übergeben werden. Über die Benutzerdaten kann man der empfangenden Task z.B. mitteilen, zu welcher TS-Anwendung die Verbindung gehört.

Die Verbindung ist nach dem *t_redrq()* in der rufenden Task nicht mehr bekannt, die Transportreferenz für diese Task ungültig. Die gerufene Task muß bereits existieren und bei der TS-Anwendung angemeldet sein; sie erhält das Ereignis T_REDIN.

Die Verbindung darf nicht umgelenkt werden,

– wenn für sie T_DATASTOP oder T_XDATSTOP vorliegt,
– während eine TIDU dieser Verbindung stückweise mit *t_datain()* (Rückgabewert: n > 0) abgeholt wird.

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

-> tref
Zeiger zu einem Feld mit der Transportreferenz der Verbindung, die umgelenkt werden soll.

<- pid
Zeiger zu einem Feld, in dem die TSN der gerufenen Task anzugeben ist.

-> opt

Für *opt* ist der Wert NULL oder der Zeiger zu einer Variante mit Benutzeroptionen anzugeben.

Damit kann man die gerufene Task bei der Verbindungsumlenkung Informationen mitschicken. Die gerufene Task nimmt diese mit der Verbindungsumlenkung entgegen. Wird *opt* = NULL angegeben, so stellt CMX(BS2000) der gerufenen Task die angegebenen Standardwerte zu.

Folgende Strukturen sind in <*cmx.h*> definiert:

```
    struct t_optc1 {
->      int  t_optnr;     /* Options-Nr. */
->      char *t_udatap;   /* Datenpuffer */
->      int  t_udatal;    /* Länge des Datenpuffers */
        int  t_xdata;     /* Vorrangdaten-Auswahl */
->      int  t_timeout;   /* Wartezeit auf Anmeldung */
    };

    struct t_optc2 {
->      int  t_optnr;     /* Options-Nr. */
->      char *t_udatap;   /* Datenpuffer */
->      int  t_udatal;    /* Länge des Datenpuffers */
    };
```

t_optnr

Optionsnummer. Anzugeben ist:

T_OPTC1 bei *t_optc1*,
T_OPTC2 bei *t_optc2*.

t_udatap

Zeiger zu einem Bereich mit Benutzerdaten, die der empfangenden Task zugestellt werden sollen.

Standardwert bei Angabe von NULL: undefiniert

t_udatal

Anzahl der aus dem Datenbereich *t_udatap* zu übertragenden Byte. Die maximal mögliche Anzahl ist in <*cmx.h*> als T_RED_SIZE definiert.
Wird *t_udatal* = 0 angegeben, so wird *t_udatap* nicht ausgewertet.

Standardwert bei Angabe von NULL: 0

t_xdata

Dieses Feld ist in dieser Version noch nicht definiert. Angaben für *t_xdata* werden ignoriert.

t_timeout
: Parameter wird von CMX(BS2000) nicht ausgewertet. Die Task, auf die die Verbindung umgelenkt werden soll, muß zum Zeitpunkt von *t_redrq()* bereits bei der TS-Anwendung angemeldet sein.

  Standardwert: T_NO.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

Hinweis

Die Speicherbereiche müssen allokiert und vom Programm lesend *(*tref, *opt, *t_udatap)* zugreifbar sein, andernfalls bricht das Programm mit Adreßfehler ab! Daß Benutzerdaten angegeben sind, erkennt CMX(BS2000) an *t_udatal* > 0, *t_udatap* = NULL ist erlaubt!

t_strerror
CMX(BS2000)-Fehlermeldung decodieren

t_strerror() decodiert CMX(BS2000)-Fehlermeldungen, die der Task beim Aufruf von *t_error()* in sedezimaler Darstellung von CMX(BS2000) übergeben wurden.

t_strerror() liefert den Zeiger auf einen statischen Bereich, der die Klartextdarstellung zu der in *code* angegebenen CMX(BS2000)-Fehlermeldung enthält.

Der Klartext setzt sich zusammen aus Fehlersymbolen, wie in <*cmx.h*> definiert, siehe unten.

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

-> code
Für *code* ist die Darstellung der Fehlermeldung anzugeben, die der Task beim Aufruf von *t_error()* von CMX(BS2000) übergeben wurde.

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_strerror()* den Zeiger auf einen Bereich mit der Klartextdarstellung der CMX(BS2000)-Fehlermeldung als String in C-Notation zurück. Der Klartext setzt sich zusammen aus den Fehlersymbolen, wie in <*cmx.h*> definiert. Jedes Fehlersymbol wird von \t eingeleitet und hat folgenden Aufbau:

"\t*typ*\n\t*klasse*\n\t*wert* DIAG-INF=0X*nnnnnnnn*\n"

Wird in *code* ein nicht definierter Wert angegeben, so liefert *t_strerror()* einen Zeiger auf folgenden Klartext:

"\t0Xnnnnnnnn ? \n"

Im Falle eines Fehlers liefert *t_strerror()* den NULL-Zeiger zurück.

Siehe auch

t_error(), t_perror()

t_strreason
Verbindungsabbaugründe decodieren

t_strreason() decodiert Verbindungsabbaugründe, die der Task beim Aufruf von *t_disin()* in sedezimaler Darstellung übergeben werden.

t_strreason() liefert einen Zeiger auf einen statischen Bereich, der die Klartextdarstellung zu dem in *reason* übergebenen Grund für den Verbindungsabbau enthält.

Der Klartext setzt sich zusammen aus dem Symbol für den Verbindungsabbaugrund, wie in <*cmx.h*> definiert, siehe unten.

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

-> reason

Für *reason* ist die Darstellung des Verbindungsabbaugrundes anzugeben, die der Task beim Aufruf von *t_disin()* von CMX(BS2000) übergeben wurde.

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_strreason()* den Zeiger auf einen Bereich mit der Klartextdarstellung des Verbindungsabbaugrundes als string in C-Notation zurück. Dieser String hat folgenden Aufbau:

"\t*reason*\n"

Wird in *reason* ein nicht definierter Wert angegeben, so liefert *t_strreason()* einen Zeiger auf folgenden Klartext:

"\t n ?\n"

Im Falle eines Fehlers liefert *t_strreason()* den NULL-Zeiger zurück.

Dateien

cmxlib.cat - Meldungsdatei

Siehe auch

t_disin(), t_preason()

t_vdatain
Daten empfangen (data indication)

t_vdatain() nimmt ein zuvor mit *t_event()* gemeldetes Ereignis T_DATAIN entgegen.

Die laufende Task übernimmt mit *t_vdatain()* auf der angegebenen Verbindung eine Transport Interface Data Unit (TIDU) der laufenden Transport Service Data Unit (TSDU) der sendenden TS-Anwendung.

t_vdatain() übernimmt die Daten einer empfangenen TIDU in mehrere nicht zusammenhängende Speicherbereiche. Diese Speicherbereiche werden mit Hilfe des Vektors *vdata* beschrieben.
Die Anzahl der Speicherbereiche, d.h. die Anzahl der Elemente in *vdata* wird im Parameter *vcnt* angegeben.
In *vdata* sind dann *vcnt* Strukturen *t_data* eingetragen. Jeder *t_data*-Eintrag beschreibt einen der Speicherbereiche vdata[0], vdata[1],..., vdata[vcnt-1].
Die empfangenen Daten werden nacheinander in diesen Speicherbereichen abgelegt. Dabei wird jeder Speicherbereich zuerst vollständig gefüllt, bevor zum nächsten Bereich übergegangen wird.
Zwischen zwei TIDUs derselben TSDU können beliebige andere CMX(BS2000)-Ereignisse für dieselbe oder eine andere Verbindung auftreten.

Die maximale Länge der TIDU ist abhängig von der verwendeten Transportverbindung. Sie kann für eine etablierte Verbindung mit *t_info()* abgefragt werden.

Keine TIDU muß vollständig gefüllt sein. Die Aufteilung einer TSDU in TIDUs ist rein lokal und erlaubt keine Rückschlüsse auf die Aufteilung der TSDU in TIDUs bei der sendenden TS-Anwendung.

t_vdatain() zeigt an:

– (im Parameter *chain*)
  ob noch eine weitere TIDU zur laufenden TSDU gehört (*chain* = T_MORE) oder nicht (*chain* = T_END).
  Die einzelnen TIDUs einer TSDU werden jeweils bei *t_event()* mit dem Ereignis T_DATAIN angezeigt.

– (mit dem Ergebniswert)
  ob die aktuelle TIDU vollständig gelesen wurde oder nicht.
  Wird der Wert T_OK zurückgegeben, so paßte die TIDU in die bereitgestellten Datenbereiche hinein. Die laufende Task hat die aktuelle TIDU vollständig übernommen.
  Wird ein Wert n > 0 zurückgegeben, so wurde nur ein Teil der TIDU gelesen. n ist die Anzahl der Byte, die aus der TIDU noch nicht gelesen wurden (Restlänge). In diesem Fall muß zunächst *t_vdatain()* oder *t_datain()* solange aufgerufen werden, bis die ganze TIDU gelesen ist. Erst dann sind wieder andere CMX(BS2000)-Aufrufe möglich, z.B. *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
Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

<> vdata
Zeiger auf einen Bereich von t_data-Strukturen für Datenpuffer, in die CMX(BS2000) die Daten der empfangenen TIDU einträgt.

Folgende Struktur ist in <*cmx.h*> definiert:

```
     struct t_data {
<-       char *t_datap;  /* Datenbereich */
< >      int t_datal;    /* Länge des Datenbereichs */
     };
```

t_datap
Zeiger auf einen Datenbereich, in den CMX(BS2000) empfangene Daten der TIDU einträgt.

t_datal
Vor dem Aufruf muß für *t_datal* die Länge des Datenbereiches *t_datap* eintragen werden (mindestens 1). Nach dem Aufruf liefert CMX(BS2000) in diesem Feld die Anzahl der eingetragenen Byte zurück.

-> vcnt
Anzahl der Elemente in *vdata*. Anzugeben ist mindestens 1 und höchstens T_VCNT.

<- flags
Zeiger zu einem Indikator, in dem CMX(BS2000) anzeigt, ob noch eine weitere TIDU zur TSDU gehört. Mögliche Werte:

T_MORE
Es folgt noch eine weitere zur TSDU gehörende TIDU. Sie wird mit einem eigenen T_DATAIN angezeigt.

T_END
Die vorliegende TIDU ist die letzte der TSDU.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich, die TIDU wurde vollständig gelesen.

n > 0
: Es sind noch n Byte in der TIDU vorhanden.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

t_vdatarq
Daten senden (data request)

t_vdatarq() sendet auf der angegebenen Verbindung die nächste (einzige) Transport Interface Data Unit (TIDU) einer Transport Service Data Unit (TSDU) an die empfangende TS-Anwendung.

Die TIDU wird in mehreren, nicht zusammenhängenden Speicherbereichen bereitgestellt. Diese Speicherbereiche werden mit Hilfe des Vektors *vdata* definiert. Die Anzahl der Speicherbereiche, d.h. die Anzahl der Elemente in *vdata* wird im Parameter *vcnt* angegeben. In *vdata* sind dann *vcnt* Strukturen *t_data* eingetragen. Jeder *t_data*-Eintrag beschreibt einen der Speicherbereiche vdata[0], vdata[1],..., vdata[vcnt-1].
CMX(BS2000) übernimmt die Daten nacheinander aus diesen Speicherbereichen. Dabei wird jeder Speicherbereich zuerst vollständig übernommen, bevor zum nächsten Bereich übergegangen wird.

Wenn die zu sendende TSDU länger ist als eine TIDU, muß sie mit mehreren Aufrufen *t_vdatarq()* (bzw. *t_datarq()*) hintereinander übermittelt werden. Deshalb kann die sendende Task bei jedem *t_vdatarq()* im Parameter *chain* angeben, ob noch eine weitere TIDU folgt, die zur selben TSDU gehört.

Die maximale Länge der TIDU hängt ab von der verwendeten Transportverbindung. Sie kann für eine etablierte Verbindung mit *t_info()* abgefragt werden.

Liefert *t_vdatarq()* das Ergebnis T_DATASTOP zurück, so ist die TIDU übernommen, der Fluß von TIDUs aber für diese Verbindung gesperrt worden.

Der Fluß der TIDUs kann gesperrt werden von:

– der empfangenden TS-Anwendung;
  sie kann den Fluß der TIDUs durch Aufruf von *t_datastop()* oder *t_xdatstop()* sperren,

– CMX(BS2000),
  wenn der lokale Zwischenspeicher ausgeschöpft ist.

Ist der Fluß der TIDUs gesperrt, so muß mit *t_event()* erst das Ereignis T_DATAGO für die Verbindung abgewartet werden, bevor die nächste TIDU gesendet werden kann.

Ein erfolgreicher Abschluß von *t_vdatarq()* (T_OK) bedeutet nicht, daß die empfangende TS-Anwendung die Daten bereits entgegengenommen hat.
Ein erfolgloser Abschluß von *t_vdatarq()* (T_ERROR) bedeutet stets, daß lokal ein Fehler erkannt wurde.

```
#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
Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

-> vdata
Zeiger auf einen Bereich von *t_data*-Strukturen für Datenpuffer, aus der CMX(BS2000) die Daten der zu sendenden TIDU entnimmt. Folgende Struktur ist in <*cmx.h*> definiert:

```
     struct t_data {
->       char *t_datap;  /* Datenbereich */
->       int t_datal;    /* Länge des Datenbereichs */
     }
```

t_datap
Zeiger auf einen Datenbereich, aus dem CMX(BS2000) zu sendende Daten der TIDU entnimmt.

t_datal
Für den Parameter ist die Länge des Datenbereiches *t_datap* anzugeben. Anzugeben ist mindestens 1 und höchstens die Länge einer TIDU. Die Summe aller t_datal-Werte darf die maximale Länge einer TIDU nicht überschreiten.

-> vcnt
Anzahl der Elemente in *vdata*. Anzugeben ist mindestens 1 und höchstens T_VCNT. Die Summe der *t_datal*-Werte aller *vcnt t_data*-Elemente darf die Länge einer TIDU nicht überschreiten.

<- flags
Zeiger zu einem Indikator, der CMX(BS2000) anzeigt, ob noch eine weitere TIDU zur TSDU gehört.
Mögliche Werte:

T_MORE
Es folgt noch eine weitere zur TSDU gehörende TIDU.

T_END
Die vorliegende TIDU ist die letzte der TSDU.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Es können sofort weitere TIDUs gesendet werden.

T_DATASTOP
: Der Aufruf war erfolgreich. Es dürfen erst weitere TIDUs gesendet werden, wenn für die angegebene Verbindung das Ereignis T_DATAGO eingetroffen ist.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

Hinweise

Es ist verboten, nach T_DATASTOP "auf Verdacht" weiterzusenden. CMX(BS2000) verhindert dies zwar nicht, Zuwiderhandelnde müssen aber damit rechnen, daß später zu viele oder bereits wieder ungültige T_DATAGO-Ereignisse eintreffen.

Der Speicherbereich **t_datap* muß allokiert und vom Programm aus lesend zugreifbar sein, sonst wird das Programm mit Adreßfehler unterbrochen. NULL ist für *t_datap* eine gültige Adresse.

t_wake()
Wecken einer Task aus t_event

Die Funktion *t_wake()* weckt die mit *pid* (=TSN) bezeichnete Task aus dem Wartepunkt *t_event()*. Die geweckte Task erhält den Returnwert T_NOEVENT. *t_wake()* ist zum Einsynchronisieren anderer Ereignisse als CMX(BS2000)-Ereignisse in den CMX(BS2000)-Wartepunkt gedacht. *t_wake()* kann eine andere Task mit der selben USER-ID oder - beim Aufruf aus einer Signalroutine - die eigene Task wecken.

t_wake() erzeugt unbedingt ein T_NOEVENT-Ereignis, auch wenn die zu weckende Task gerade nicht *t_event()* aufruft.

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

-> pid
Zeiger zu einem Feld mit der TSN der zu weckenden Task.

-> evref
Der Wert von evref muß immer NULL sein.

Rückgabewert

T_OK
Der Aufruf war erfolgreich.

T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_event(), t_error()

t_xdatgo
Vorrangdatenanzeige freigeben (expedited data go)

t_xdatgo() gibt die gesperrte Vorrangdatenanzeige auf der angegebenen Verbindung frei. Die laufende Task teilt CMX(BS2000) damit mit, daß sie wieder bereit ist, Vorrangdaten entgegenzunehmen.

Der Aufruf bewirkt, daß der laufenden Task wieder das Ereignis T_XDATIN für die angegebene Verbindung zugestellt wird, sofern dieses vorliegt.

Der Aufruf *t_xdatgo()* ist nur zulässig, wenn beim Aufbau dieser Verbindung der Austausch von Vorrangdaten vereinbart wurde.

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

-> tref
: Zeiger zu einem Feld mit der Transportreferenz der Verbindung, auf der die Vorrangdatenanzeige wieder freigegeben werden soll.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_event(), t_error(), t_xdatstop()

Hinweis

In CMX(BS2000) ist die Datenflußkontrolle am Interface entkoppelt von der Datenflußkontrolle auf der Transportverbindung (bedingt durch BCAM). Das heißt, *t_xdatstop()* - *t_xdatgo()* wird von der Partner-TS-Anwendung erst dann bemerkt, wenn die Flußkontrolle der Transportverbindung reagiert.

t_xdatin
Vorrangdaten empfangen (expedited data indication)

t_xdatin() nimmt ein zuvor mit *t_event()* gemeldetes Ereignis T_XDATIN entgegen. Dabei muß *t_xdatin()* vor dem nächsten *t_event()* aufgerufen werden.

Die laufende Task übernimmt mit *t_xdatin()* auf der angegebenen Verbindung eine Expedited Transport Service Data Unit (ETSDU) von der sendenden TS-Anwendung. Die maximale Länge der ETSDU ist abhängig von der verwendeten Transportverbindung. Sie ist aber nie größer als T_EXP_SIZE Byte.

Wenn die Vorrangdaten in den bereitgestellten Bereich *datap* hineinpassen, wird der Wert T_OK zurückgegeben. Andernfalls wird ein Wert > 0 zurückgegeben. n ist die Anzahl der Byte, die aus der ETSDU noch nicht gelesen wurden (Restlänge). In diesem Fall muß zunächst *t_xdatin()* solange aufgerufen werden, bis die ganze ETSDU gelesen ist. Erst dann können wieder andere CMX(BS2000)-Aufrufe abgesetzt werden, z.B. *t_event()*.

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

-> tref
: Zeiger zu einem Feld mit der bei *t_event()* erhaltenen Transportreferenz der Verbindung.

<- datap
: Zeiger auf einen Bereich, in den CMX(BS2000) die Daten der empfangenen ETSDU einträgt.

<> datal
: Zeiger zu einem Feld, in das vor dem Aufruf die Länge des Datenbereiches *datap* eintragen werden muß. Anzugeben ist mindestens 1.
Nach dem Aufruf liefert CMX(BS2000) in diesem Feld die Anzahl der eingetragenen Byte zurück.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Die Vorrangdaten wurden vollständig gelesen.

n > 0
: Es sind noch n Byte in der ETSDU vorhanden.

T_ERROR
: Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_error(), t_event()

t_xdatrq
Vorrangdaten senden (expedited data request)

t_xdatrq() sendet auf der angegebenen Verbindung eine Expedited Transport Service Data Unit (ETSDU) mit Vorrangdaten an die empfangende TS-Anwendung. Die maximale Länge der ETSDU hängt ab von der verwendeten Transportverbindung. Sie ist aber nie größer als T_EXP_SIZE Byte.

Der Aufruf *t_xdatrq()* ist nur dann zulässig, wenn beim Aufbau der entsprechenden Verbindung der Austausch von Vorrangdaten vereinbart wurde.

ETSDUs können vorher abgeschickte Transport Interface Data Units (TIDUs) mit Normaldaten überholen. Es ist garantiert, daß ETSDUs nie später bei der empfangenden TS-Anwendung eintreffen als TIDUs, die nach ihnen abgeschickt wurden.

Bei Rückmeldung T_XDATSTOP ist die ETSDU übernommen, der Fluß von ETSDUs und TIDUs aber für diese Verbindung gesperrt worden.

Der Vorrangdatenfluß kann gesperrt werden von:

– der empfangenden TS-Anwendung;
  sie kann den Fluß der ETSDUs durch Aufruf von *t_xdatstop()* sperren,

– CMX(BS2000),
  wenn der lokale Zwischenspeicher voll ist.

Ist der Fluß der ETSDUs gesperrt, so muß mit *t_event()* erst das Ereignis T_XDATGO oder T_DATAGO für die Verbindung abgewartet werden, bevor weitere ETSDUs gesendet werden können.

Ein erfolgreicher Abschluß von *t_xdatrq()* (T_OK) bedeutet nicht, daß die empfangende TS-Anwendung die Daten bereits entgegengenommen hat.
Ein erfolgloser Abschluß von *t_xdatrq()* (T_ERROR) bedeutet stets, daß lokal ein Fehler erkannt wurde.

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

-> tref
  Zeiger zu einem Feld mit der Transportreferenz der Verbindung, auf der die Vorrangdaten gesendet werden sollen.

-> datap
Zeiger auf einen Bereich mit der zu sendenden ETSDU.

-> datal
Zeiger zu einem Feld mit der Anzahl der zu sendenden Byte aus dem dem Bereich *datap*.

Minimalwert: 1

Maximalwert: T_EXP_SIZE
(T_EXP_SIZE ist in <*cmx.h*> definiert.)

Rückgabewert

T_OK
Der Aufruf war erfolgreich, weitere Vorrangdaten können sofort gesendet werden.

T_XDATSTOP
Der Aufruf war erfolgreich, weitere ETSDUs dürfen erst gesendet werden, wenn für diese Verbindung eines der Ereignisse T_XDATGO oder T_DATAGO eingetroffen ist.

T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

t_error(), t_event(), t_xdatstop()

t_xdatstop
Vorrangdatenanzeige sperren (expedited data stop)

t_xdatstop() sperrt die Anzeige für Vorrang- und Normaldaten auf der angegebenen Verbindung.

t_xdatstop() bewirkt im einzelnen:

– Die laufende Task teilt CMX(BS2000) dadurch mit, daß sie bis auf weiteres nicht bereit ist, für diese Verbindung Normal- oder Vorrangdaten zu empfangen. Ein bereits angezeigtes Ereignis T_DATAIN oder T_XDATIN muß aber erst beantwortet werden.

– Die laufende Task bekommt die Ereignisse T_XDATIN und T_DATAIN für die angegebene Verbindung nicht mehr zugestellt. Sie kann aber, während die Datenanzeige gesperrt ist, andere CMX(BS2000)-Funktionen aufrufen, z.B. eine weitere Verbindung aufbauen, abbauen oder umlenken.

– Die sendende TS-Anwendung darf solange noch senden, bis die Datenflußregelung auf der Transportverbindung reagiert.

– Die sendende TS-Anwendung erhält (im Verlauf) bei *t_xdatrq()* das Ergebnis T_XDATSTOP bzw. bei *t_datarq()* das Ergebnis T_DATASTOP. Sie darf keine Normal- oder Vorrangdaten mehr senden.

Freigegeben wird die Vorrangdatenanzeige mit *t_xdatgo()* oder mit *t_datago()*.

Der Aufruf *t_xdatstop()* ist nur zulässig, wenn beim Aufbau dieser Verbindung der Austausch von Vorrangdaten vereinbart wurde.

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

-> tref
Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

Rückgabewert

T_OK
Der Aufruf war erfolgreich.
T_ERROR
Fehler. Fehlercode mit *t_error()* abfragen.

Fehler

Im Fehlerfall mögliche Fehlerwerte können durch Aufruf von *t_error()* abgefragt werden. Im Anhang befindet sich eine Liste aller Fehlerwerte.

Siehe auch

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

9 Programmbeispiele

Die folgenden beiden Beispiele zeigen eine Anwendung (rcopy) zur Übertragung von Textdateien. Die Anwendung ist aufgeteilt in Client- (Beispiel 1) und Server-Programm (Beispiel 2). In den Beispielen ist die Verwendung von case-Konstrukten zur Auswertung der CMX-Ereignisse und die Aufteilung von beliebig langen TSDUs in TIDUs (probeweise tiduln fix auf 32 setzen!) dargestellt. Im Beispiel 2 ist die Zuordnung (Verbindung - Datei) nur über *t_ucepid* realisiert.

Beispiel 1

```
#include <stdio.h>                  /* Standard Bibliothek */
#include <cmx.h>                    /* CMX Bibliothek */

/*————————————————————————————— 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);
}
```

Beispiel 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 = {T_OPTI1};
static struct t_optc3 optc = {T_OPTC3,NULL,0,T_YES,T_NO,0};
static struct t_opte1 opte = {T_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);
}
```

Das folgende Beispiel veranschaulicht die Einträge in den Name Service.

Beispiel 3

Wird der Server mit dem Namen "zentrale" auf dem Rechner **HOSTWIEN** und der client "cmx002" auf dem Rechner **D016ZE01** gestartet, dann müssen vorher folgende /BCMAP-Einträge vom BCAM-Administrator abgesetzt werden:

auf **D016ZE01**:

```
/BCMAP FU=DEF,SUBFU=LOCAL,APPL=(OSI,C'cmx002'),TSEL-I=(5,C'RCOPY')
/BCMAP FU=DEF,SUBFU=GLOBAL,APPL=(OSI,C'zentrale'),ES=HOSTWIEN,PTSEL-I=(8,C'RCOPYSRV')
```

auf **HOSTWIEN**:

```
/BCMAP FU=DEF,SUBFU=LOCAL,APPL=(OSI,C'zentrale'),TSEL-I=(8,C'RCOPYSRV')
/BCMAP FU=DEF,SUBFU=GLOBAL,APPL=(OSI,C'cmx002'),ES=D016ZE01,PTSEL-I=(5,C'RCOPY')
```

Die T-Selektoren können beliebig vergeben werden, müssen aber innerhalb eines Rechners eindeutig sein. Es empfiehlt sich, die Einträge in einer Datei zu hinterlegen.

10 Anhang

10.1 Vergleich CMX(BS2000) mit CMX(SINIX)

Aufgrund der unterschiedlichen Systemarchitektur gibt es in ICMX(BS2000) Differenzen zu ICMX(SINIX). Diese Differenzen betreffen vor allem jene, die CMX-Programme zwischen SINIX und BS2000 portieren wollen. Die Regeln des Zustandsautomaten müssen auch bei CMX(BS2000) strikt eingehalten werden!

Im BS2000 nicht ausgewertete Parameter

- CMX(BS2000) überwacht keine anwendungsspezifischen Verbindungslimits. Die Parameter *t_apmode* und *t_conlim* sind beim *t_attach()* immer auf T_ACTIVE + T_PASSIVE + T_REDIRECT bzw. T_NOLIMIT eingestellt. Ankommende Verbindungswünsche erhält immer die älteste Task der TS-Anwendung. Die Funktionalität von *t_apmode* und *t_conlim* kann durch geeignete *t_disrq()* bzw. *t_redrq()*-Aufrufe ersetzt werden.
- CMX(BS2000) überwacht nicht die Inaktivzeit von Verbindungen. Der Parameter *t_timeout* wird bei *t_conrq()* und *t_conrs()* immer mit T_NO angenommen.
- Die Task, auf die eine Verbindung umgelenkt werden soll, muß zum Zeitpunkt der Umlenkung bereits existieren und an die TS-Anwendung angemeldet sein. Bei *t_redrq()* wird der Parameter *t_timeout* immer mit T_NO angenommen.

Unterschiede im Verhalten von CMX(BS2000) zu CMX(SINIX)

- Bei CMX(BS2000) enthalten die Errorcodes zusätzlich BS2000-spezifische Diagnoseinformation. Bei Fehlertypen > 15 kann mit error = error & 0xff der CMX-Fehlerwert extrahiert werden. Es ist nicht garantiert, daß bei gleichen Fehlersituationen in BS2000 und SINIX der gleiche Errorwert geliefert wird.
  Besonders T_WTREF bekommt bei CMX(BS2000) eine zusätzliche Bedeutung:
  1. Die tref ist falsch.
  2. Ein Aufruf bezog sich auf eine Verbindung, die gerade vom Partner abgebaut wurde - das Ereignis T_DISIN folgt. (entspricht in SINIX T_COLLISION).

- Ein Wartezustand beim *t_event()*-Aufruf wird im BS2000 nicht automatisch durch eine Signalroutine beendet. Dafür steht der Aufruf *t_wake()* zur Verfügung.

Unterschiede im Transportsystem

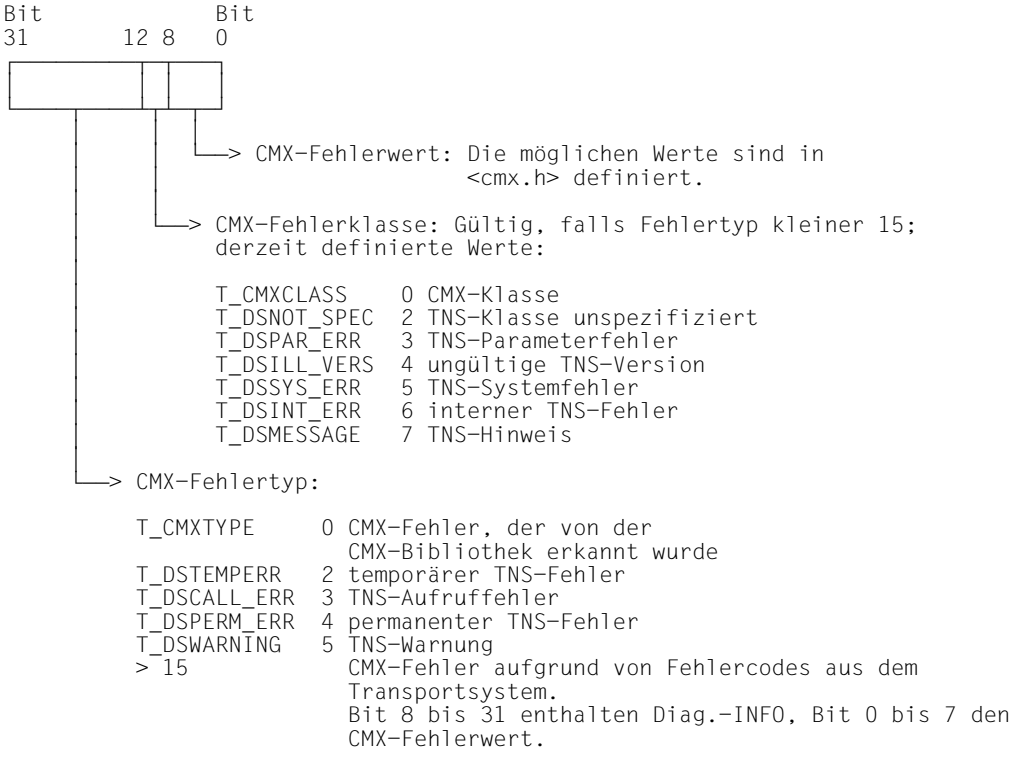
- Der TNSX ist im BS2000 durch den BCAM-Nameservice (/BCMAP-Kommando) realisiert. Hier müssen ebenfalls alle globalen Namen eingetragen werden
- Die Maximalanzahl der TS-Anwendungen und Verbindungen kann im BS2000 per BCAM-Administrationskommandos in weiten Grenzen eingestellt werden - ebenso die Wartezeit auf Verbindungsaufforderungen.
- Im BS2000 ist es möglich, eine TS-Anwendung per BCAM-Administrationskommando zu schließen. Dieser in SINIX (noch) nicht vorgesehene Fall wird von CMX(BS2000) V1.0 bei *t_event* mit T_ERROR, Errorwert = T_CCP_END angezeigt. Als Reaktion darauf sollten alle TS-Anwendungen des Programmes geschlossen und neu eröffnet bzw. das Programm beendet werden.

10.2 CMX(BS2000)-Fehlermeldungen

Die folgenden Tabellen enthalten alle möglichen CMX(BS2000)-Fehlermeldungen, d.h. alle Fehlermeldungen, die an der Programmschnittstelle ICMX erzeugt werden. Die Fehlermeldungen sind nach Fehlertyp und Fehlerklasse sortiert.

Aufbau des CMX(BS2000)-Returnwertes

Jede Fehlermeldung an ICMX wird übergeben in der Form: 0x%x. %x ist dabei ein 32 Bit langer Fehlercode. Der Fehlercode ist wie folgt aufgebaut.



Die Funktion *t_strerror()* bildet aus dem CMX-Fehlerwert den Ausgabestring:

"\ttyp\n\tklasse\n\twert DIAG-INF = 0xnnnnnnnn \n"

Die Funktion *p_error()* gibt diesen String auf Standardfehlerausgabe *stderr* aus.

Aus dem Fehler-Wert der anderen CMX-Funktionen kann z.B. mit "error = t_attach() & 0xff" ein kompatibles Verhalten zu SINIX erreicht werden.

Die CMX-Fehlerwerte und deren Bedeutung finden Sie auf der nächsten Seite. Die Diagnosecodes (=BCAM-Returncodes), die in DIAG-INF ausgegeben werden, sind samt Zuordnung und Bedeutung in den darauf folgenden Tabellen aufgelistet.

Fehlerwerte von CMX

num. Wert	symbolischer Wert	Bedeutung
0	T_NOERROR	Kein Fehler
5	T_EIO	momentaner Engpaß bzw. Fehler im Transportsystem
14	T_EFAULT	IO-Area nicht allokiert
100	T_UNSPECIFIED	Nicht näher spezifizierter Fehler, i.a. Fehler bei einem Systemaufruf.
101	T_WSEQUENCE	Ungültige Aufruf-Reihenfolge
103	T_WPARAMETER	Fehlerhafter Parameter
104	T_WAPPLICATION	Die Anwendung ist unbekannt oder die Task ist nicht zur Anmeldung der Anwendung berechtigt oder die Anwendung ist von dieser Task bereits eröffnet.
105	T_WAPP_LIMIT	Es ist keine Anmeldung von Tasks in Anwendungen mehr möglich; der Grenzwert für gleichzeitig aktive Anwendungen ist erreicht.
106	T_WCONN_LIMIT	Grenzwert für gleichzeitig aktive Verbindungen erreicht
107	T_WTREF	Unzulässige Transportreferenz oder Transportverbindung bereits abgebaut.
111	T_NOCCP	Das Transportsystem unterstützt die gewünschte Anmeldung oder Verbindung nicht.
114	T_CCP_END	Das Transportsystem (BCAM) wurde beendet oder die Anwendung wurde vom Administrator geschlossen.
255	T_WLIBVERSION	Kein Anschluß an das CMX-Subsystem möglich.

Zuordnung der BCAM-Returncodes zu den CMX-Fehlerwerten

C M X - Fehler-Wert	**DIAG - INF**	**a t t**	**c c f**	**c i n**	**c r q**	**c r s**	**d g o**	**d i n**	**d r q**	**d s t**	**d e t**	**d s n**	**d s q**	**e r r**	**e v e**	**i n f**	**r i n**	**r r q**	**v i n**	**v r q**	**w a k**	**x g o**	**x i n**	**x r q**	**x s t**
	S-RTC 2 1 / M-RTC 2 1																								
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 - Fehler-Wert	**DIAG - INF**	**att**	**ccf**	**cin**	**crq**	**crs**	**dgo**	**din**	**drq**	**dst**	**det**	**dsn**	**dsq**	**err**	**eve**	**inf**	**rin**	**rrq**	**vin**	**vrq**	**wak**	**xgo**	**xin**	**xrq**	**xst**
	S-RTC 2 1 **M-RTC 2 1**																								
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 - Fehler-Wert	DIAG - INF S-RTC 2 1	DIAG - INF 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

Bedeutung der Diagnosecodes

C M X - Fehler- Wert	DIAG-Info S-RTC 2 1	DIAG-Info M-RTC 2 1	Bedeutung
T_EFAULT	00 00	31 08	User Buffer nicht zugreifbar
T_EIO	00 00	00 28	Der Aufruf kann im Moment nicht durchgeführt werden (Aufruf später wiederholen)
T_EIO	04 00	01 1C	Kein Speicher für Datenpuffer vorhanden
T_EIO	04 00	02 1C	Keine freie Transportreferenz vorhanden
T_EIO	04 00	03 1C	Kein Speicher für ACONCB vorhanden
T_EIO	04 00	04 1C	Kein Speicher für APPCB vorhanden
T_EIO	04 00	06 1C	Kein Speicher für SUB-TCB vorhanden
T_EIO	04 00	08 1C	BS2000 Börse überladen
T_EIO	04 00	09 1C	Kein Speicher für ENACB vorhanden
T_EIO	04 00	0A 1C	kein Speicher für ADDRCB-P vorhanden
T_EIO	04 00	0B 1C	Keine freie CONNECTION_ID vorhanden
T_EIO	04 00	0C 1C	Kein Speicher für Layer 4 CB vorhanden
T_EIO	04 00	0D 1C	Keine freie APID vorhanden
T_EIO	04 00	0E 1C	Keine freie Portnummer vorhanden
T_EIO	04 00	10 1C	Kein Speicher für Lokalen Ereignisgruppen Kontrollblock vorhanden
T_EIO	04 00	11 1C	Kein Speicher für Globalen Ereignisgruppen Kontrollblock vorhanden
T_EIO	04 00	13 1C	Kein Nameserver Entry vorhanden
T_EIO	04 00	14 1C	Kein Speicher für Ereignissgruppennamen vorhanden
T_EIO	04 00	15 1C	Kein Speicher für EVOL vorhanden
T_CCP_END	08 00	01 1C	BCAM Shutdown angekündigt
T_CCP_END	0C 00	01 1C	BCAM Quick Shutdown
T_EIO	10 00	00 1C	Globaler Grenzwert für die Anzahl der eröffneten TSAPs erreicht
T_EIO	00 00	01 50	Unbekannter Host
T_EIO	00 00	02 50	Host nicht aktiv
T_EIO	00 00	03 50	Ungültige eigene INTERNET_ADDRESS
T_EIO	00 00	01 30	System Error beim Anstarten des CONHAND processings
T_EIO	00 00	02 30	System Error beim Warten auf die Beendigung des CON-HAND processings
T_EIO	00 00	00 10	Keine (Vorrang-)Daten eingetroffen
T_EIO	00 00	00 2C	Benutzerdaten verloren gegangen
T_EIO	00 00	01 2C	Verbinungsdaten verloren gegangen
T_EIO	08 00	00 10	Kein Telegramm vorhanden
T_NOCCP	0C 00	00 20	TSAP von einer anderen Task exklusiv eröffnet
T_NOCCP	10 00	00 20	TSAP bereits von dieser Task eröffnet
T_NOCCP	18 00	00 24	Partner nicht bekannt
T_NOCCP	1C 00	00 20	TSAP nicht aktiv
T_NOCCP	1C 00	00 24	Partner Prozessor nicht bekannt

C M X - Fehler- Wert	DIAG-Info S-RTC 2 1	DIAG-Info M-RTC 2 1	Bedeutung
T_NOCCP	1C 00 01 24		Partner Prozessor nicht aktiv
T_NOCCP	1C 00 02 24		Route(n) nicht bekannt
T_NOCCP	1C 00 03 24		Route(n) nicht aktiv
T_NOCCP	1C 00 04 24		Partner IP Adresse nicht bekannt
T_NOCCP	1C 00 06 24		Verbindungsaufbauwunsch zu Broadcastadesse
T_NOCCP	20 00 00 20		*t_conrq* für TSAP nicht erlaubt
T_NOCCP	2C 00 00 20		TSAP Password nicht gültig
T_NOCCP	30 00 00 20		TSAP konnte nicht neu eröffnet werden
T_NOCCP	38 00 00 20		TSAP wäre neu eröffnet worden
T_NOCCP	3C 00 00 20		Zugriff auf TSAP über CMX nicht erlaubt
T_NOCCP	40 00 00 24		Die angegebene Verbindung erlaubt das Senden von Vorrangdaten nicht
T_NOCCP	40 00 04 24		Verbindungsdaten nicht erlaubt
T_NOCCP	40 00 05 24		Geforderte Interfacefunktionalität wird nicht unterstützt
T_NOCCP	40 00 05 24		Geforderte Interfacefunktionalität nicht unterstützt
T_NOCCP	40 00 07 24		Interfacefunktionalität der Partner paßt nicht zusammen
T_NOCCP	40 00 08 24		Level 4 Addresse nicht vorhanden
T_NOCCP	04 00 01 1C		Kein Speicher für Datenpuffer vorhanden
T_WAPPLICATION	00 00 01 51		Ereignisgruppe von anderer Task bereits shareable eröffnet
T_WAPPLICATION	00 00 02 51		Ereignisgruppe bereits von dieser Task eröffnet
T_WAPPLICATION	00 00 03 51		Ereignisgruppe von anderer Task bereits exclusive eröffnet
T_WAPPLICATION	00 00 04 51		Task nicht an der Ereignisgruppe angeschlossen
T_WAPPLICATION	00 00 06 51		Standard Ereignisgruppe nicht exclusiv eröffnet
T_WAPPLICATION	00 00 08 51		Standard Ereignisgruppe nicht eröffnet
T_WAPPLICATION	00 00 0B 51		Keine tasklokale Ereignisgruppe angegeben
T_WAPPLICATION	00 00 0C 51		Aufrufer hat nicht die gleiche USERID wie der Eigentümer der Ereignisgruppe
T_WAPPLICATION	00 00 0D 51		Die verbindungspezifischen Ereignisse werden nicht an eine Ereignissgruppe gemeldet
T_WAPPLICATION	00 00 0E 51		Die TSAP-spezifischen Ereignisse werden nicht an eine Ereignissgruppe gemeldet
T_WAPPLICATION	00 00 10 51		USERID des Eigentümers der Ereignisgruppe konnte nicht ermittelt werden
T_WAPPLICATION	00 00 12 51		EVENT_ID der Ereignisgruppe ist ungültig
T_WAPPLICATION	00 00 13 51		Ereignisgruppe kann im Moment nicht geschlossen werden, da sie noch in Benutzung ist
T_WAPPLICATION	04 00 00 20		TSAP nicht von dieser Task eröffnet
T_WAPPLICATION	18 00 01 20		TSAP wird gerade zwangsweise durch den BCAM Administrator geschlossen
T_WAPPLICATION	40 00 00 20		Privilegien zum Eröffnen des TSAPs nicht vorhanden
T_WAPPLIMIT	28 00 00 20		Tasklokaler Grenzwert für die Anzahl der eröffneten TSAPs erreicht

C M X - Fehler- Wert	DIAG-Info S-RTC 2 1	DIAG-Info M-RTC 2 1	Bedeutung
T_WCONNLIMIT	24 00 00 20		Keine weiteren Verbindungen für diesen TSAP erlaubt
T_WPARAMETER	00 00 00 14		Benutzerdatenlänge zu groß
T_WPARAMETER	00 00 00 14		Länge der Vorrang- bzw. Verbindungsdaten zu groß
T_WPARAMETER	00 00 02 08		Fehlerhafte Syntax des NEA TSAP Namens
T_WPARAMETER	00 00 03 08		TSAPOPEN_ID nicht angegeben
T_WPARAMETER	00 00 07 08		CONNECTION_ID nicht angegeben
T_WPARAMETER	00 00 0A 08		LENGTH_OF_USER_BUFFER ungültig
T_WPARAMETER	00 00 13 08		User Buffer length = 0
T_WPARAMETER	00 00 16 08		Vorrangdatenlänge = 0
T_WPARAMETER	00 00 18 08		NUMBER_OF_USER_BUFFER nicht angegeben
T_WPARAMETER	00 00 1B 08		Ungültige Anzahl von Routennamen
T_WPARAMETER	00 00 1F 08		NEA TSAP-Name nicht angegeben
T_WPARAMETER	00 00 20 08		Kein ISO TSAP-Name angegeben
T_WPARAMETER	00 00 21 08		Länge des TSAP-Namens nicht angegeben
T_WPARAMETER	00 00 26 08		LENGTH_OF_USER_BUFFER_2 ungültig
T_WPARAMETER	00 00 27 08		Ungültige WAKE_TSN
T_WPARAMETER	00 00 29 08		Partner TSAP-Name nicht angegeben
T_WPARAMETER	00 00 2A 08		Kein ISO Partner TSAP-Name angegeben
T_WPARAMETER	00 00 2B 08		Länge des Partner TSAP-Namens nicht angegeben
T_WPARAMETER	00 00 2C 08		Keine Vorrangdaten vorhanden
T_WPARAMETER	00 00 2D 08		Keine Normaldaten vorhanden
T_WPARAMETER	00 00 2E 08		TYPE_OF_INFORMATION ungültig
T_WPARAMETER	00 00 2F 08		TYPE_OF_TRANSFER_INDICATION ungültig
T_WPARAMETER	00 00 30 08		ungültiger Socket Hostname
T_WPARAMETER	00 00 33 08		Angegebene Portnummer wird bereits genutzt
T_WPARAMETER	00 00 34 08		LENGTH_OF_USER_BUFFER_1 ungültig
T_WPARAMETER	00 00 35 08		Angegebene Adressinformation widerspricht der durch / BCMAP Kommando definierten
T_WPARAMETER	00 00 36 08		LEVEL_3_CONNECTION_USER_DATA falsch
T_WPARAMETER	00 00 37 08		Endsystem-Name (Parameter-Angabe oder /BCMAP Kommando definiert) widerspricht der angegebenen IP-Adresse
T_WPARAMETER	00 00 00 28		*t_redrq* kann im Moment nicht durchgeführt werden (Aufruf später wiederholen)
T_WSEQUENCE	00 00 00 3C		Die angegebene Verbindung erlaubt das Empfangen bzw. Senden von Vorrangdaten nicht
T_WSEQUENCE	04 00 02 24		Nutzung von Expedited Data nicht erlaubt
T_WSEQUENCE	08 00 00 24		Verbindung bereits aufgebaut
T_WSEQUENCE	0C 00 00 24		Verbindung wird bereits aufgebaut
T_WSEQUENCE	14 00 02 24		Kein Connection Request anstehend
T_WSEQUENCE	14 00 00 1C		Warten auf DATA_GO_INDICATION
T_WSEQUENCE	14 00 00 1C		Warten auf EXPDATA_GO_INDICATION
T_WSEQUENCE	30 00 00 24		TSAP nicht zum Verbindungsaufbau berechtigt

C M X - Fehler- Wert	DIAG-Info S-RTC 2 1	DIAG-Info M-RTC 2 1	Bedeutung
T_WSEQUENCE	48 00 00 24		Verbindung ist nicht in der Datentransferphase (noch nicht komplett aufgebaut)
T_WSEQUENCE	60 00 00 24		PORT Nummer wird bereits genutzt
T_WSEQUENCE	64 00 00 24		Verbindung wird bereits abgebaut
T_WSEQUENCE	68 00 00 24		Angegebene NEA-Adresse wird bereits von einem anderen TSAP genutzt.
T_WTREF	04 00 00 24		Ungültige CONNECTION_ID

10.3 Liste der Verbindungsabbaugründe

Im folgenden sind die von CMX(BS2000) bei den Aufrufen *t_disin()* und *x_disin()* in *reason* übergebenen Verbindungsabbaugründe beschrieben. Die hier angegebenen symbolischen Werte sind in <*cmx.h*> numerisch definiert. "Lokales Transportsystem" steht für das Transportsystem im System der laufenden Task, "Partner-Transportsystem" für das Transportsystem im System des Verbindungspartners der laufenden Task.

Von CMX(BS2000) angegebene Gründe:

num. Wert	symbolischer Wert	Bedeutung
0	T_USER	Der Abbau erfolgte durch den Kommunikationspartner; u.U auch durch einen Benutzerfehler des Kommunikationspartners.
1	T_RTIMEOUT	Wegen Inaktivität der Verbindung gemäß Parameter t_timeout wurde die Verbindung lokal durch CMX abgebaut.
2	T_RADMIN	Wegen Außerbetriebnahme des Transportsystems durch die Administration wurde die Verbindung lokal durch CMX abgebaut.
3	T_RCCPEND	Wegen Transportsystem-Ausfall wurde die Verbindung lokal durch CMX abgebaut.

Vom Partner-Transportsystem angegebene Gründe:

num. Wert	symbolischer Wert	Bedeutung
256	T_RUNKNOWN	Der Partner oder das Transportsystem hat die Verbindung abgebaut. Ein Grund für den Abbau wurde nicht angegeben.
257	T_RSAPCONGEST	Wegen eines TSAP-spezifischen Engpasses hat das Partner-Transportsystem die Verbindung abgebaut.
258	T_RSAPNOTATT	Das Partner-Transportsystem hat die Verbindung abgebaut, weil der adressierte TSAP dort nicht angemeldet ist.
259	T_RUNSAP	Das Partner-Transportsystem hat die Verbindung abgebaut, weil der adressierte TSAP dort nicht bekannt ist.
261	T_RPERMLOST	Abbau durch Netzadministration oder Administration des Partner-Transportsystem.
262	T_RSYSERR	Fehler im Netz.
385	T_RCONGEST	Wegen Betriebsmittelengpaß hat das Partner-CCP die Verbindung abgebaut.
386	T_RCONNFAIL	Wegen Mißlingen des Verbindungsaufbaus hat das Partner-Transportsystem die Verbindung abgebaut. Der Verbindungsaufbau kann mißlingen, weil z.B. Benutzerdaten zu lang oder Vorrangdaten nicht zugelassen sind.
387	T_RDUPREF	Weil für ein NSAP-Paar eine zweite Verbindungsreferenz vergeben wurde (Systemfehler), wurde die Verbindung vom Partner-Transportsystem abgebaut.
388	T_RMISREF	Wegen einer nicht zuzuordnenden Verbindungsreferenz (Systemfehler) hat das Partner-Transportsystem die Verbindung abgebaut.
389	T_RPROTERR	Wegen eines Protokollfehlers (Systemfehler) hat das Partner-Transportsystem die Verbindung abgebaut.
391	T_RREFOFLOW	Wegen Verbindungsreferenz-Überlaufs hat das Partner-Transportsystem die Verbindung abgebaut.
392	T_RNOCONN	Das Partner-Transportsystem hat den Aufbau der Netzverbindung abgelehnt.
394	T_RINLNG	Wegen falscher Header- oder Parameterlänge (Systemfehler) hat das Partner-Transportsystem die Verbindung abgebaut.

Vom lokalen Transportsystem angegebene Gründe:

num. Wert	symbolischer Wert	Bedeutung
448	T_RLCONGEST	Wegen Betriebsmittelengpaß hat das lokale Transportsystem die Verbindung abgebaut.
449	T_RLNOQOS	Weil Quality of Service nicht mehr geboten werden kann, hat das lokale Transportsystem die Verbindung abgebaut.
451	T_RILLPWD	Ungültiges (Verbindungs-) Passwort.
452	T_RNETACC	Netzzugang wurde verweigert.
464	T_RLPROTERR	Wegen eines Transportprotokollfehlers (Systemfehler) hat das lokale Transportsystem die Verbindung abgebaut.
465	T_RLINTIDU	Weil es eine zu lange Schnittstellen-Dateneinheit (TIDU) erhalten hat (Systemfehler), hat das lokale Transportsystem die Verbindung abgebaut.
466	T_RLNORMFLOW	Wegen Verletzung der Flußkontrollregeln für Normaldaten (Systemfehler) hat das lokale Transportsystem die Verbindung abgebaut.
467	T_RLEXFLOW	Wegen Verletzung der Flußkontrollregeln für Vorrangdaten (Systemfehler) hat das lokale Transportsystem die Verbindung abgebaut.
468	T_RLINSAPID	Weil es eine ungültige TSAP-Identifikation erhalten hat (Systemfehler), hat das lokale Transportsystem die Verbindung abgebaut.
469	T_RLINCEPID	Weil es eine ungültige TCEP-Identifikation erhalten hat (Systemfehler), hat das lokale Transportsystem die Verbindung abgebaut.
470	T_RLINPAR	Wegen eines unzulässigen Parameterwerts, z.B. Benutzerdaten zu lang oder Vorrangdaten nicht zugelassen, hat das lokale Transportsystem die Verbindung abgebaut.
480	T_RLNOPERM	Die Administration des lokalen Transportsystem hat den Verbindungsaufbau verhindert.
481	T_RLPERMLOST	Die Administration des lokalen Transportsystem hat die Verbindung abgebaut.
482	T_RLNOCONN	Weil keine Netzverbindung verfügbar ist, konnte das lokale Transportsystem den Verbindungsaufbau nicht durchführen.
483	T_RLCONNLOST	Wegen Verlust der Netzverbindung hat das lokale Transportsystem die Verbindung abgebaut.
484	T_RLNORESP	Weil der Partner nicht auf CONRQ antwortet, ist der Verbindungsaufbau vom lokalen Transportsystem nicht durchführbar.
485	T_RLIDLETRAF	Wegen Verlust der Verbindung (Idle Traffic Timeout) hat das lokale Transportsystem die Verbindung abgebaut.
486	T_RLRESYNC	Weil die Resynchronisierung erfolglos war (mehr als 10 Wiederholungen), hat das lokale Transportsystem die Verbindung abgebaut.

num. Wert	symbolischer Wert	Bedeutung
487	T_RLEXLOST	Weil der Vorrangdatenkanal defekt ist (mehr als 3 Wiederholungen), hat das lokale Transportsystem die Verbindung abgebaut.

Fachwörter

Adresse
siehe *TRANSDATA-Adresse* und *TRANSPORTADRESSE*.

Aktiver Partner
Der *Kommunikationpartner*, der selbst eine *Verbindung* zu einer anderen TS-Anwendung aufbaut.

Anwendung
Siehe *TS-Anwendung*.

ASCII
Internationaler Zeichensatz für DV-Systeme auf 7 Bit-Basis (ISO-7Bit-Code).

CCITT
Organisation öffentlicher Netzwerkbetreiber und Postverwaltungen, Genf.

Dateneinheit
Die Zeichenmenge, die man mit einem Aufruf t_datarq() auf einmal senden oder mit einem Aufruf t_datain() empfangen kann.

DCAM-Anwendung
Eine *TS-Anwendung* im BS2000, die die Zugriffsmethode DCAM benutzt.

EBCDIC
EBCDIC-Code ist ein auf 8 Bit erweiterter BCD-Code, der auf BS2000-Rechnern, TRANSDATA-Kommunikationsrechnern und IBM-kompatiblen Maschinen verwendet wird.

ETSDU
Vorrangdateneinheit.

GLOBALER NAME
Name einer *TS-Anwendung*, der sie im Netz eindeutig identifiziert. Unter dem GLOBALEN NAMEN steht eine *TS-Anwendung* im *TS-Directory.*

ICMX

Standard-Transportsystem-Schnittstelle für Anwendungen.

ISO-Referenzmodell

Modell für die Kommunikation 'Offener Systeme'. Es ist in der Norm ISO 7498 beschrieben und enthält 7 Schichten.

KOGS

Spezielle Sprache, um Netzwerk-Konfigurationen zu beschreiben.

Kommunikationsmethode

Eine Zugriffsmethode, die auf die Transportdienste des *ISO-Referenzmodells* aufsetzt.

Kommunikationspartner

Eine *TS-Anwendung*, die eine logische Verbindung zu einer anderen *TS-Anwendung* unterhält und Daten mit ihr austauscht.

LOKALER NAME

Eigenschaft einer *TS-Anwendung* im *TS-Directory* zum *GLOBALEN NAMEN*. Der LOKALE NAME muß bei der Anmeldung bei CMX(BS2000) angegeben werden.

Nachricht

Eine logisch zusammengehörige Datenmenge, die an einen *Kommunikationspartner* gesendet werden soll.

Partner

siehe *Kommunikationspartner*.

Passiver Partner

Ein *Kommunikationspartner*, der eine *Verbindung* nicht selbst aufbaut, sondern von einem anderen Kommunikationspartner angesprochen wird.

Prozessor

Netzweit adressierbare Instanz im Verarbeitungsrechner oder Kommunikationsrechner, in der die Leistungen des Transportservices erbracht werden.

Prozessorname

Ein Teil der *TRANSDATA-Adresse*. Der Prozessorname wird angegeben in der Form:
prozessornummer/regionsnummer.

TIDU

Dateneinheit

TRANSPORTADRESSE

Eigenschaft einer *TS-Anwendung* im *TS-Directory* zum *GLOBALEN NAMEN*. Die TRANSPORTADRESSE muß beim Verbindungsaufbau zu einem Kommunikationspartner angegeben werden. Sie besitzt als Wert die von CMX(BS2000) geforderte Transportadresse.

Transport Name Service in SINIX und SINIX-ODT

Dienst in CMX(SINIX) zur Verwaltung transportsystemspezifischer Eigenschaften von *TS-Anwendungen*.

Transportreferenz

Eine Nummer, die innerhalb einer *TS-Anwendung* eine *Verbindung* eindeutig kennzeichnet.

Transportschicht

1. Schicht im *ISO-Referenzmodell*. Sie wird beschrieben durch die Norm ISO 8072.

Transportsystem

Die unteren 4 Schichten im *ISO-Referenzmodell*.

TSAP

Zugriffspunkt für eine *TS-Anwendung* auf das Transportsystem.

TS-Anwendung

Transportservice-Anwendung:
Eine TS-Anwendung ist eine Anwendung, die die Dienste des Transportsystems nutzt. Sie besteht aus Programmen, die eine logische *Verbindung* zu einer anderen TS-Anwendung aufbauen können, um mit dieser Daten auszutauschen.

TSDU

Nachricht

Verbindung, logische

Zuordnung zweier *Kommunikationspartner*, die es ihnen ermöglicht, Daten miteinander auszutauschen.

Abkürzungen

ASCII	American Standard Code of 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 Decimals Interchange Code
EMDS	Emulation Datensichtstation
EOF	End of File
EOS	End of String
ETSDU	Expedited Transport Service Data Unit
FT	File Transfer
ICMX	Interface Communication Method SINIX und SINIX Open Desktop
ISO	International Organization for Standardization
KOGS	Konfigurationsorientierte Generatorsprache
KR	Kommunikationsrechner
LAN	Local Area Network
NEA	Netzwerk-Architektur bei TRANSDATA-Systemen
OSI	Open System Interconnection

PDN	Programmsystem für Datenfernverarbeitung und Netzsteuerung
TCEP	Transport Connection Endpoint
TIDU	Transport Interface Data Unit
TS	Transport Service
TSAP	Transport Service Access Point
TSDU	Transport Service Data Unit
VAR	Verarbeitungsrechner
WAN	Wide Area Network

Literatur

ISO-Normen-Bezugsquelle:

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

Wenden Sie sich zum Bestellen von Handbüchern bitte an Ihre zuständige Geschäftsstelle.

Stichwörter

C

D

U

V

Inhalt

CMX (BS2000) V1.0A

Kommunikationsmethode im BS2000

Zielgruppe
Programmierer von Transport-Service-Anwendungen (TS-Anwendungen)

Inhalt
CMX (BS2000) bietet Anwendungsprogrammen eine einheitliche Schnittstelle zu den Transportdiensten. Mit CMX (BS2000) können Sie Anwendungsprogramme erstellen, die unabhängig vom Transportsystem mit anderen Anwendungen kommunizieren können.

Ausgabe: September 1992

Datei: 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.