Betriebssystem SINIX
CMX

Kommunikationsmethode in SINIX

Benutzerhandbuch

Ausgabe Februar 1988 (CMX V2.1)

Bestell-Nr. U2405-J-Z95-4
Printed in the Federal Republic of Germany
6470 AG 2881.5 (8080)

Siemens Aktiengesellschaft

Vorwort

Dieses Manual richtet sich an Programmierer von Kommunikationsanwendungen in SINIX. Solche Anwendungen bestehen aus in C implementierten Anwendungsprogrammen, die mit Kommunikationspartnern in einem Netzwerk von PCs, Kommunikations- und Verarbeitungsrechnern (mit Einschränkungen auch in Fremdsystemen) Nachrichten austauschen. Die Kommunikation folgt den internationalen Normungen im OSI-Schichtenmodell. In SINIX stehen dem Programmierer über die Kommunikationsmethode CMX (Communication Method SINIX) die erforderlichen Programmschnittstellen zur Verfügung.

Vom Programmierer wird die Beherrschung der Programmiersprache C und des C-Entwicklungssystems [CES] in SINIX [SIN1] erwartet. Ferner ist für das Verständnis das Wissen über die Prinzipien und Methoden der Datenfernverarbeitung hilfreich, insbesondere über das OSI-Schichtenmodell wie in DIN ISO 7498 normiert.

Kapitel 1	gibt eine Einführung in die Funktionen von CMX.
Kapitel 2 bis 4	enthalten Hinweise zur Programmierung mit CMX, eingeteilt nach den Kommunikationsphasen Anmeldung, Verbindungsaufbau und Datenaustausch.
Kapitel 5	beschreibt die Verwendung der CMX-Funktionen zum Transport Name Service für die Adressierung.
Kapitel 6	beschreibt die CMX-Aufrufe in alphabetischer Reihenfolge.
Kapitel 7	beschreibt das NEABV-Protokoll.
Kapitel 8	umfaßt Hinweise zur Installation von CMX und zur Abhängigkeit von anderen Produkten sowie zum Übersetzen, Binden und zur Diagnose einer Kommunikationsanwendung.

Manualredaktion K D ST QM 2
Otto-Hahn-Ring 6, 8000 München 83

Änderungsprotokoll 1

Änderung des Vorgänger-Manuals,
Stand Juli 1987 (CMX V2.0),
durch die Neuausgabe vom
Februar 1988 (CMX V2.1)

Das gesamte Manual wurde überarbeitet. Insbesondere sei auf folgendes hingewiesen:

- Kapitel 7 ist neu. Es wird darin das NEABV-Protokoll beschrieben. Die bisherigen Abschnitte des Kapitels 7 finden Sie jetzt in Kapitel 8.
- Das Programm cmxt in Abschnitt 8.6 beinhaltet jetzt auch die bisherigen Programme cxlt und cxwt, deren Beschreibung gelöscht wurde.
- Im Abschnitt 8.6 wurde das bisherige Programm dstrace in tnsxt umbenannt.
- Neu hinzu kam in Abschnitt 8.6 das Programm tnsxprop. Es dient zur Ausgabe der Attribute eines Objektes eines TS-Directory.
- Der Abschnitt 8.7 Verfolgerinformationen für NEABX ist neu.
- Im Anhang wurden die Include-Dateien cmx.h, tnsx.h und neabx.h durch aktuelle Dateien ersetzt.

Inhalt

1 CMX ist die Kommunikationsmethode in SINIX

Kommunizieren heißt: zwei Partner tauschen miteinander Informationen aus. Sie müssen sich dabei einer von beiden Seiten akzeptierten Methodik unterwerfen. In der Kommunikationstechnologie ist diese Methodik durch internationale Normung vorgegeben, die im OSI-Schichtenmodell aufgeschrieben ist.

Die Kommunikationspartner tauschen ihre Informationen unter Verwendung von Transportsystemen (die unteren vier Schichten des OSI-Schichtenmodells) aus. Diese übernehmen die Vermittlung und den Transport der Daten über die physikalischen Medien: Leitungen, Rechner, Netzwerke. Die Dienste der Transportsysteme sind im OSI-Schichtenmodell in der Norm ISO 8072 festgelegt. In SINIX-Systemen werden die Dienste der Transportsysteme über CMX angeboten. CMX ist also die Kommunikationsmethode in SINIX.

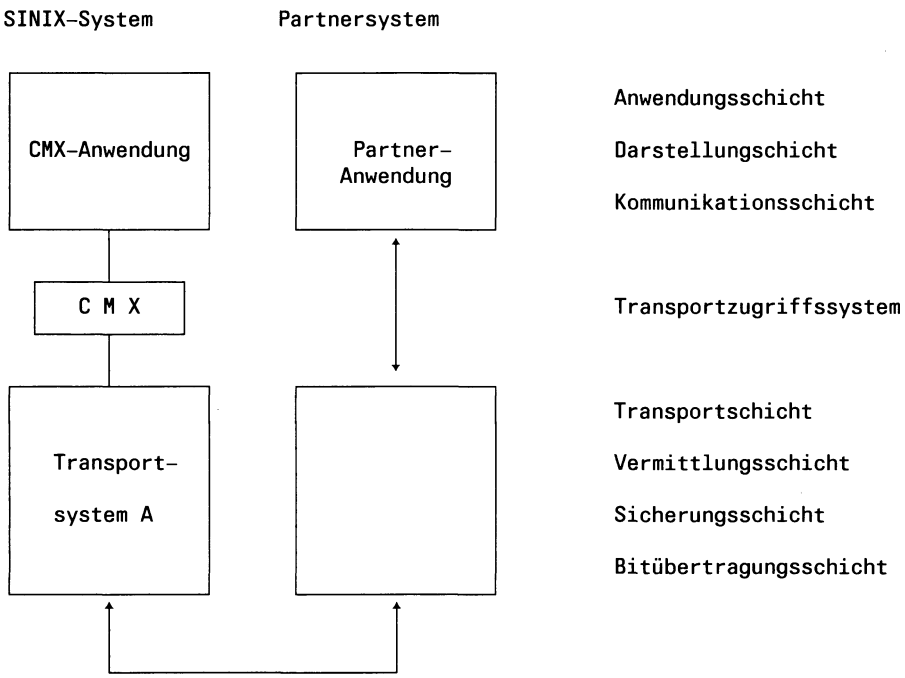


Bild 1-1 CMX ist eine Kommunikationsmethode

CMX enthält eine Programmschnittstelle für die Verwendung in C- und COBOL-Programmen in SINIX. Damit kann man Anwendungen schreiben, die miteinander kommunizieren können. CMX bietet dem Programmierer die Dienste verschiedener Transportsysteme in einheitlicher Weise an. Für den Programmierer bedeutet es, daß er unabhängig ist von spezifischen Eigenschaften der Datenübermittlung, wie Sicherung der Übertragung, Vermittlung, usw.. Er programmiert nur die CMX-Funktionen, die dazu dienen:

- seine CMX-Anwendung bei CMX anzumelden,
- Transportverbindungen zu Kommunikationspartnern aufzubauen,
- Daten zu senden und zu empfangen,
- den Datenfluß zu regeln.

1.1 Wer kann mit wem kommunizieren?

SINIX bietet laufend erweiterte Vernetzungsmöglichkeiten für SINIX-Rechner durch Fort- und Neuentwicklung von Transportsystemen für die verschiedenen Netzwerktechnologien und Protokollstandards. Für alle neuen Transportsysteme dient CMX als Kommunikationsmethode. Die Kommunikationslandschaft für eine CMX-Anwendung weitet sich also stetig aus, immer mehr und neue Kommunikationspartner werden erreichbar.

CMX unterstützt in einheitlicher Weise die Kommunikation von Kommunikationspartnern in SINIX-Rechnern mit solchen in anderen SINIX-Rechnern, in Kommunikationsrechnern (KR) mit PDN und Verarbeitungsrechnern (VAR) mit BS2000 innerhalb des TRANSDATA-Netzes. Kommunikationspartner in Fremdsystemen sind erreichbar, sofern sie die im OSI-Modell aufgeschriebenen Normen erfüllen.

Die für SINIX-Rechner möglichen Ankopplungsarten an Netzwerke werden in folgende Klassen unterteilt:

Stationskopplung STA an Anschlußrechner mit PDN

Die Anbindung des SINIX-Rechners an das Netzwerk erfolgt als Station(en) an einem Anschlußrechner, üblicherweise ein KR der TRANSDATA-Familie mit dem Betriebssystem PDN. Jede im SINIX-Rechner residierende CMX-Anwendung ist aus der Sicht des Netzes eine Datenstation am Anschlußrechner. Die Transportsysteme im SINIX-Rechner enthalten nur die Bitübertragungs- und Sicherungsschicht, Vermittlungs- und Transportschicht bleiben leer. Das eigentliche Transportsystem liegt im Anschlußrechner. Es übernimmt die Weitervermittlung tiefer ins Netz hinein.

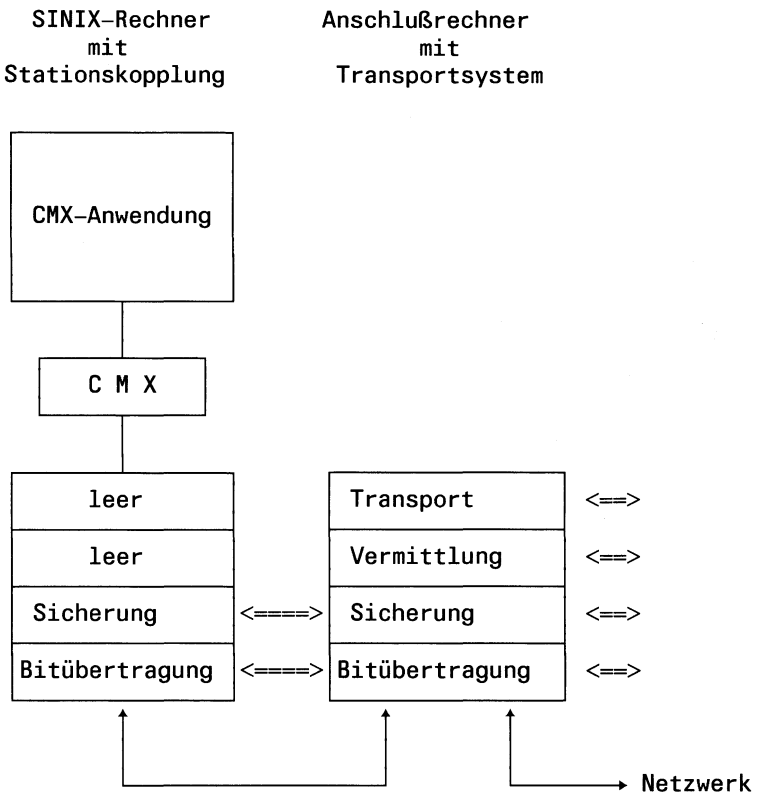


Bild 1-2 Stationskopplung mit CMX

Die Transportverbindung endet im Anschlußrechner, zur CMX-Anwendung führt eine Lokalverbindung, wie in TRANSDATA für Stationen üblich. Der Partner adressiert diese Station am Anschlußrechner. Der CMX-Anwendung stehen nicht alle Dienste des Transportsystems im Anschlußrechner zur Verfügung.

Rechnerkopplung an WAN ("wide area network")

Die Anbindung des SINIX-Rechners ist hier für öffentliche Netzwerke vorgesehen.
Im SINIX-Rechner residiert ein voll ausgebildetes Transportsystem, das die im WAN erforderlichen Protokolle realisiert. Es werden verschiedene Protokolle unterstützt. Solche können die im TRANSDATA-Netz vereinbarten sein, aber auch international genormte Protokolle gemäß dem OSI-Schichtenmodell.

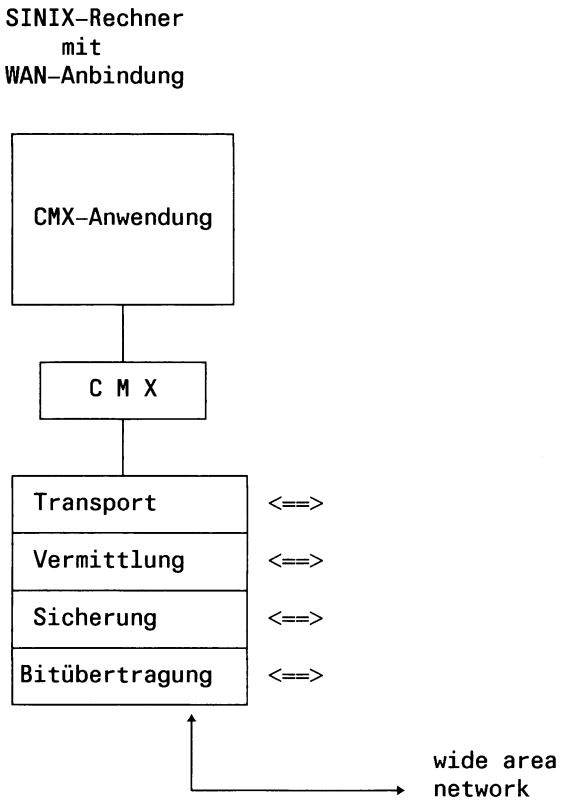


Bild 1-3 Rechnerkopplung an WAN mit CMX

Bei Rechnerkopplung an WAN stehen der CMX-Anwendung alle Dienste des Transportsystems zur Verfügung.

Rechnerkopplung an LAN ("local area network")

Die Anbindung des SINIX-Rechners erfolgt hier an im allgemeinen private Netzwerke innerhalb eines Gebäudes oder eines Standortes, also geographisch begrenzt. Im SINIX-Rechner residiert ein voll ausgebildetes Transportsystem, das die im LAN verwendeten, international normierten Protokolle gemäß OSI-Schichtenmodell realisiert.

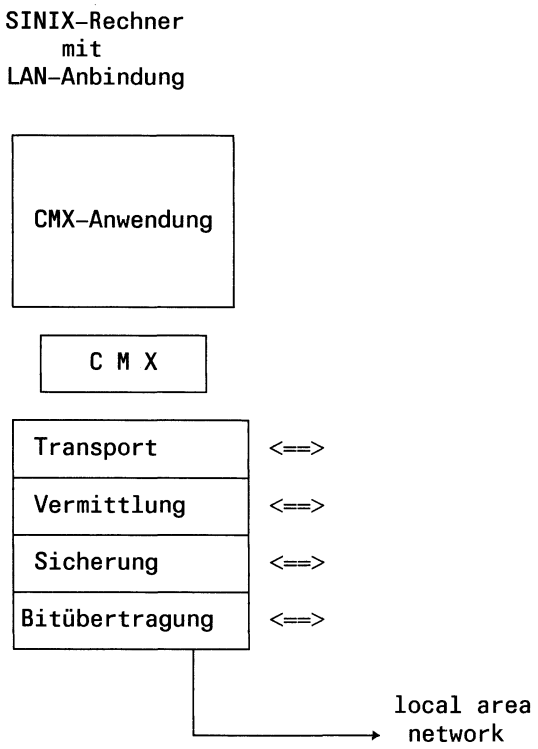


Bild 1-4 Rechnerkopplung an LAN mit CMX

Bei Rechnerkopplung an LAN stehen der CMX-Anwendung alle Dienste des Transportsystems zur Verfügung.

In ein und demselben Rechnernetzwerk können zugleich mehrere der Ankopplungsarten vorhanden sein. In einem SINIX-Rechner sind je nach Modell mehrere der Ankopplungsarten gleichzeitig einsetzbar.

Gemeinsam ist allen Ankopplungsarten, daß sie von derselben Kommunikationsmethode CMX bedient werden.

1.2 Funktionen von CMX - eine Übersicht

Neben den eigentlichen Funktionen zur Kommunikation, wie Verbindungsbehandlung und Datenaustausch, bietet CMX zwei wichtige Dienste. Der erste dient zur Verwaltung der Namen und Adressen von CMX-Anwendungen und heißt

Transport Name Service (TNS).

Der andere erlaubt es CMX-Anwendungen, mit Kommunikationspartnern im TRANSDATA-Netz, die an die OSI-Konventionen nicht angepaßt sind, zu kommunizieren. Er heißt

NEABX-Migrationsservice.

Die Dienste stehen über die folgenden Schnittstellen zur Verfügung:

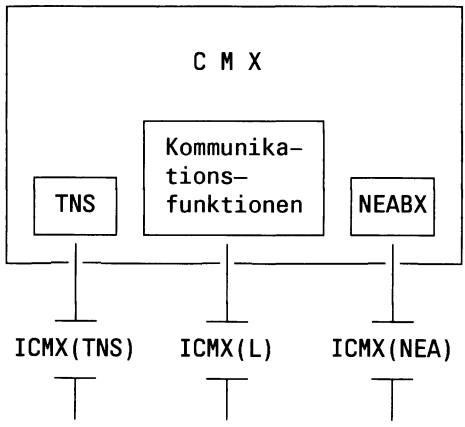


Bild 1-5 CMX-Schnittstellen

1.2.1 CMX-Funktionen zum Transport Name Service (TNS)

Jedes Netzwerk und jedes Transportsystem verlangt seine eigene spezifische Methode zur Adressierung der Kommunikationspartner. Mit dem TNS bietet CMX einen Dienst, die Partneradressen aus einem Verzeichnis, dem **TS-Directory** (Transport Service Directory), zu lesen. Sie sind dort unter logischen Namen, den "GLOBALEN NAMEN" des Kommunikationspartners abgelegt. Die GLOBALEN NAMEN sind im TS-Directory in Form eines Namensbaumes mit Wurzel, Knoten und Blättern organisiert. Ein GLOBALER NAME beschreibt den Pfad von der Wurzel dieses Baumes zu einem Knoten oder Blatt. Die Komponenten des Pfades heißen Namensteile des GLOBALEN NAMENS. Der GLOBALE NAME identifiziert den Kommunikationspartner in einer Form, die besonders für die Verwendung durch menschliche Benutzer geeignet ist.

Eine CMX-Anwendung kennt ihre Kommunikationspartner unter deren GLOBALEN NAMEN, sie selbst ist ebenfalls über ihren GLOBALEN NAMEN im Netz identifizierbar. Sie brauchen also nur die GLOBALEN NAMEN zu kennen und zu verwenden. Zum eigenen GLOBALEN NAMEN ermitteln CMX-Anwendungen ihren "LOKALEN NAMEN" mittels des TNS. Der LOKALE NAME ist ein Parameter für die Arbeit mit CMX. Zum GLOBALEN NAMEN des Kommunikationspartners ermittelt die CMX-Anwendung vor dem Verbindungsaufbau dessen "TRANSPORTADRESSE" mittels des TNS. Die TRANSPORTADRESSE ist ein Parameter beim Verbindungsaufbau. Sie brauchen sich weder um Format oder um Inhalt dieser beiden Parameter zu kümmern, Sie übergeben die Bytestrings direkt als Parameter an CMX weiter.

Bei einem ankommenden Verbindungsaufbau liefert CMX die TRANSPORTADRESSE des Kommunikationspartners. Sie übersetzen sie mit Hilfe des TNS in den GLOBALEN NAMEN des Kommunikationspartners.

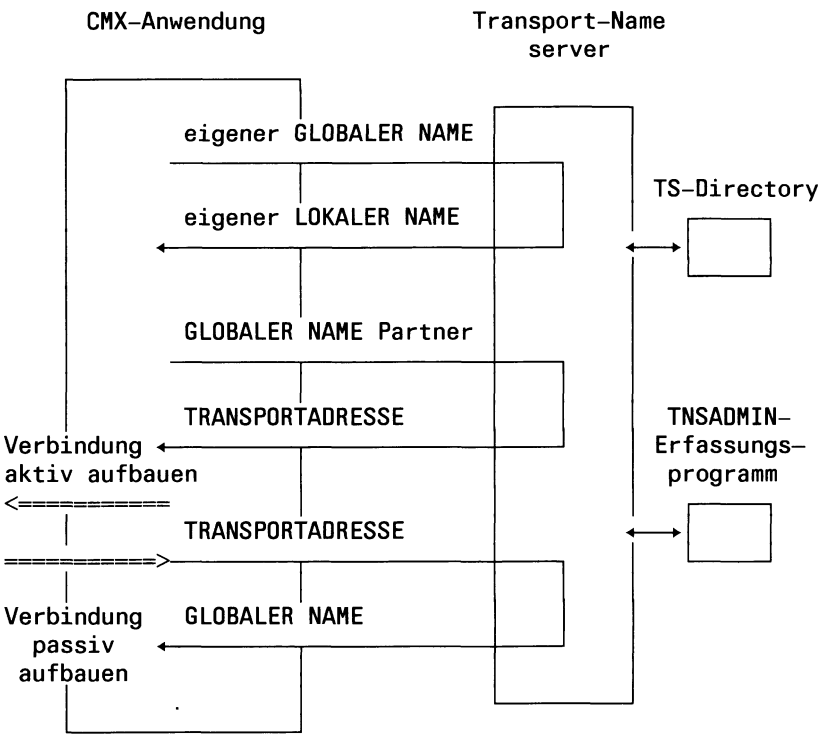


Bild 1-6 Verwendung von TNS-Funktionen in CMX-Anwendungen

Eine CMX-Anwendung braucht sich also nur mit GLOBALEN NAMEN zu beschäftigen, die Umwandlung in die Bit und Byte für das Transportsystem und das Netzwerk erfolgt außerhalb der CMX-Anwendung im TNS.

Die Beschickung des TS-Directory mit den netzwerkspezifischen Informationen zu den GLOBALEN NAMEN erfolgt durch das TNS-Erfassungsprogramm TNSADMIN. Das ist ein Werkzeug außerhalb von CMX und im Manual zu den Transportsystemen [CCP] beschrieben.

Der TNS bietet sowohl den CMX-Anwendungen Dienste zur Abfrage von Informationen als auch dem TNSADMIN Dienste zur Verwaltung und Pflege solcher Informationen im TS-Directory.

Die CMX-Funktionen zum TNS setzen sich zusammen aus Funktionen zur Informationsabfrage (siehe Abschnitt 5.1) und zur Informationsverwaltung (siehe Abschnitt 5.2). Die Informationsabfrage braucht **jede** CMX-Anwendung. Die Informationsverwaltung dient zur Pflege des TS-Directory. Sie kann von jeder CMX-Anwendung verwendet werden. Das oben erwähnte Erfassungsprogramm TNSADMIN nutzt die Informationsverwaltung aus.

Informationsabfrage

Jede CMX-Anwendung hat einen GLOBALEN NAMEN, unter dem sie im Netz bekannt ist. Bei der Anmeldung bei CMX zur Kommunikation muß sie ihren LOKALEN NAMEN als Parameter übergeben. Der LOKALE NAME kennzeichnet die CMX-Anwendung eindeutig im lokalen Endsystem. Beim Verbindungsaufbau zum Kommunikationspartner muß eine CMX-Anwendung als Parameter den LOKALEN NAMEN angeben, unter dem sie sich angemeldet hat, sowie die TRANSPORT-ADRESSE des Kommunikationspartners. Zur Übersetzung von GLOBALEN NAMEN in LOKALE NAMEN bzw. TRANSPORT-ADRESSE und umgekehrt dienen die beiden Funktionen:

Abfrage Eigenschaften zu GLOBALEN NAMEN - ReadProperties
Ermittlung GLOBALE NAMEN zu Eigenschaft - ReadLeafs

Die dritte Funktion zur Informationsabfrage dient der Ermittlung vollständiger GLOBALER NAMEN, wenn nur Teile davon bekannt sind. Sie bietet die Möglichkeit, mit teilqualifizierten GLOBALEN NAMEN zu arbeiten:

Expansion teilqualifizierter GLOBALER NAMEN - ReadChildren

Informationsverwaltung

Eine beliebige Anwendung, die Inhalt oder Struktur des TS-Directory verwalten will, bedient sich dazu der Funktionen zur Informationsverwaltung. Ein zum System gehörendes Programm für diesen Zweck ist TNSADMIN [CCP].

Einem Blatt im Namensbaum können Eigenschaften zugordnet werden, z.B LOKALER NAME und TRANSPORTADRESSE. Weitere Eigenschaften sind MIGRATIONSSERVICE, ROUTINGINFORMATION, TRANSPORTSYSTEM und GERAETETYP. Sie sind in Abschnitt 2.1 erläutert. Die Funktionen zur Informationsverwaltung heißen:

Einrichten Knoten im TS-Directory	- CreateNonLeafEntity
Löschen Knoten im TS-Directory	- DeleteNonLeafEntity
Einrichten Blatt im TS-Directory	- CreateLeafEntity
Löschen Blatt im TS-Directory	- DeleteLeafEntity
Zuordnen Eigenschaft	- AddProperty
Löschen Eigenschaft	- DeleteProperty
Ändern Eigenschaft	- ChangeProperty

1.2.2 CMX-Funktionen zur Kommunikation (CMX_L)

An- und Abmelden bei CMX

Jede CMX-Anwendung muß sich als erstes bei CMX anmelden (Abschnitt 2.2):

Anmelden Prozeß bei CMX - attach
Abmelden Prozeß bei CMX - detach

Dabei ist der LOKALE NAME der CMX-Anwendung nötig, für die die An- oder Abmeldung erfolgen soll. Sie ermitteln ihn mit den in Abschnitt 5.1 beschriebenen TNS-Funktionen zur Informationsabfrage.

Verbindungsaufbau

Dazu gehören die Funktionen zum aktiven Verbindungsaufbau (Abschnitt 3.1):

Verbindung aufbauen - connection request
Verbindung bestätigen - connection confirmation

und zum passiven Verbindungsaufbau

Verbindungsaufbauanzeige annehmen - connection indication
Verbindungsaufbau beantworten - connection response

Im Abschnitt 3.1 ist beschrieben, wie diese Aufrufe zusammenhängen. Beim aktiven Verbindungsaufbau ist die TRANSPORTADRESSE des Kommunikationspartners erforderlich. Sie ermitteln sie zum GLOBALEN NAMEN des Kommunikationspartners mit den oben beschriebenen TNS-Funktionen zur Informationsabfrage. Beim passiven Verbindungsaufbau, insbesondere bei dessen Entgegennahme, erhalten Sie die TRANSPORT-ADRESSE des Kommunikationspartners. Aus dieser ermitteln Sie mit den TNS-Funktionen dessen GLOBALEN NAMEN.

Verbindungsabbau

Zum Abbau einer Verbindung bzw. zur Entgegennahme eines Abbauwunsches gibt es (Abschnitt 3.2):

Verbindung abbauen - disconnect request
Verbindungsabbauanzeige annehmen - disconnect indication

Verbindung umlenken

Innerhalb einer CMX-Anwendung kann eine Verbindung an einen anderen Prozeß derselben Anwendung weitergegeben (umgelenkt) werden (Abschnitt 3.4):

Verbindung umlenken - redirect request
Verbindungsumlenkung annehmen - redirect indication

Datenübertragung

Mit diesen Funktionen überträgt man beliebige Daten zu einem Kommunikationspartner (Abschnitt 4.1):

Daten senden - data request
Daten empfangen - data indication

Vorrangdatenübertragung (optional)

Mit diesen Funktionen können Sie kleine Datenmengen mit Vorrang zum Hauptstrom der Daten zu einem Kommunikationspartner übertragen (Abschnitt 4.2):

Vorrangdaten senden - expedited data request
Vorrangdaten empfangen - expedited data indication

Flußregelung

Wenn Sie gerade keine Daten empfangen wollen oder können, teilen Sie dies CMX mit den folgenden Funktionen mit. CMX zeigt dann keine Daten mehr an, dem Kommunikationspartnerpartner wird dies (in der Regel) mitgeteilt, er darf dann nichts mehr senden. Der Datenfluß kann getrennt für Daten und Vorrangdaten geregelt werden (Abschnitt 4.3):

Datenfluß stoppen - datastop
Datenfluß freigeben - datago
Vorrangdatenfluß stoppen - xdatstop
Vorrangdatenfluß freigeben - xdatgo

Anfragen bei CMX

Mit diesen Funktionen kann man bei CMX Informationen abfragen (Abschnitt 2.4):

Ereignis abwarten oder abholen - event
Fehler abfragen - error
Information über CMX-Parameter - information

Ein CMX-Ereignis ist z.B. der Verbindungswunsch eines Kommunikationspartners. Die Behandlung von Ereignissen ist im Abschnitt 2.4 beschrieben.

1.2.3 CMX-Funktionen zur Migration - NEABX-Service

Die CMX-Funktionen zur Migration unterstützen die Kommunikation von Kommunikationspartnern in SINIX-Rechnern mit solchen in Kommunikationsrechnern mit PDN und Verarbeitungsrechnern mit BS2000 innerhalb des TRANSDATA-Netzes, soweit diese Funktionen nutzen, die bisher in den NEA-Transportprotokollen zur Verfügung standen, in ISO-Transportsystemen nach ISO-Norm 8072 aber nicht mehr angeboten werden.

Dieser erweiterte Funktionsumfang wird im TRANSDATA-Netz durch das NEABX-Protokoll geboten. Aufbau und Auswertung dieses Protokolls erfolgt durch die CMX-Funktionen zur Migration über die Schnittstelle ICMX(NEA), die Funktionsaufrufe erfolgen in analoger Weise zu den unter 1.2.2 beschriebenen CMX-Funktionen zur Kommunikation. Der NEABX-Migrationsservice ist damit ein Dienst, der einer bestehenden Anwendung im TRANSDATA-Netz die Kommunikation mit einer CMX-Anwendung in SINIX ermöglicht. Die Anwendung im TRANSDATA-Netz kann daher ohne Änderung ihrer Kommunikationsschnittstelle mit einer CMX-Anwendung kommunizieren. Umgekehrt ermöglicht der NEABX-Migrationsservice einer CMX-Anwendung, die erweiterte Funktionalität einer bestehenden Anwendung im TRANSDATA-Netz zu bedienen.

Mit NEABX-steht also ein Migrationsservice zur Verfügung, der für die Kommunikation zwischen CMX-Anwendungen und bereits vorhandenen Anwendungen im TRANSDATA-Netz so lange notwendig ist, bis die Kommunikationsschnittstellen der Anwendungen im Netz an die Funktionalität des ISO-Transportsystems angepaßt sind.

Entscheidungskriterien für den Einsatz

Die Entscheidung für den Einsatz des NEABX-Migrationsservices muß in der CMX-Anwendung vor dem Anmelden bei CMX getroffen werden. Der Einsatz ist immer dann zwingend, wenn der gewünschte Kommunikationspartner Funktionen benötigt, die im NEABX-Protokoll geboten werden, in einem Transportdienst entsprechend der ISO-Norm 8072 (CMX) aber nicht verfügbar sind. Diese Funktionen sind:

- Paßwort beim Verbindungsaufbau
- Benutzerdaten beim Verbindungsaufbau länger als 32 Byte
- Nachrichtenstrukturierung mit ETX/ETB
- Mitteilung des verwendeten Nachrichtencodes
- Anforderung von Transportquittungen
- Sequenznummern beim Nachrichtenaustausch

Der Einsatz des NEABX-Migrationsservices ist insbesondere dann erforderlich, wenn Sie mit heutigen BS2000-Anwendungen über DCAM, TIAM, UTM kommunizieren wollen.

Eine CMX-Anwendung kann zu jedem Zeitpunkt immer nur einen Modus verwenden. Entweder sie kommuniziert über die CMX-Schnittstelle ICMX(L) oder über die CMX-Schnittstelle zur Migration ICMX(NEA). SINIX-Prozesse, die gleichzeitig in beiden Modi kommunizieren wollen, müssen sich daher als zwei verschiedene CMX-Anwendungen unter unterschiedlichen lokalen Namen und anderen Eigenschaften bei CMX anmelden (Abschnitt 2.1).

CMX-Funktionen mit NEABX-Migrationsservice

Die CMX-Aufrufe zur Migration und der Ablauf der Kommunikation sind im wesentlichen die gleichen wie bei den CMX-Funktionen zur Kommunikation. Die Beschreibung dieser Funktionen im Manual und die Programmbeispiele treffen daher auf beide Funktionsklassen gleichermaßen zu. Auf die bestehenden Unterschiede wird jeweils besonders hingewiesen. Die Funktionsaufrufe sind im Abschnitt 6.3 im Detail beschrieben.

1.2.4 Optionen des Systems und des Benutzers

Der Funktionsumfang von CMX besteht aus obligatorischen und optionalen Funktionen mit obligatorischen und optionalen Parametern.

Für die Kommunikation mit Partnern über CMX stehen bei allen Transportsystemen stets die obligatorischen Funktionen mit den obligatorischen Parametern zur Verfügung.

Abhängig von der verwendeten Ankopplungsart, i.w. also vom Transportsystem, 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	b
Verbindungslimit Aktiv/Passiv-Modus	n	j	b

s= Systemoption, b = Benutzeroption, n/j = nein/ja

Tabelle 1-1 Optionen von CMX

Die Systemoptionen richten sich nach dem Funktionsumfang des Transportsystems. Wenn Sie Optionen verwenden, die das Transportsystem oder die Kommunikationsschnittstelle der Partneranwendung nicht bietet, kommt der Verbindungsaufbau nicht zustande, oder Sie erhalten eine Verbindungsabbauanzeige von CMX. Bei Bereitstellung eines geeigneten Transportsystems garantiert CMX den einwandfreien Ablauf Ihrer CMX-Anwendung.

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

Das bedeutet, CMX gleicht nicht die Differenz zwischen der in Ihrer CMX-Anwendung erwarteten und der vom Transportsystem tatsächlich gebotenen Funktionalität aus. Dies gilt insbesondere für die obigen Systemoptionen.

Welche Systemoptionen ein bestimmtes Transportsystem bietet ist in [CCP] beschrieben (siehe auch Abschnitt 8.4).

1.3 Die drei Schnittstellen von CMX

Die drei Schnittstellen von CMX

ICMX(L) - zur eigentlichen Kommunikation
ICMX(TNS) - zum Transport Name Service
ICMX(NEA) - zum Migrationsservice

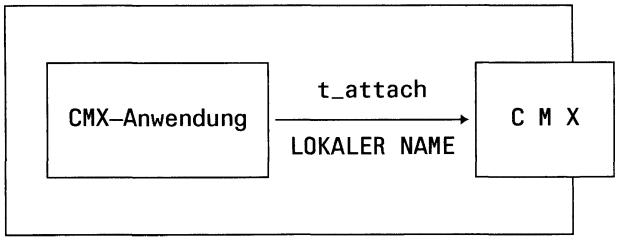
sind alle Bibliotheksschnittstellen. Das heißt, in der Bibliothek **/lib/libcmx.a** stehen die Funktionen in Modulen zur Verfügung, die zu den Programmen der CMX-Anwendung hinzugebunden werden.

1.3.1 Was ist eine CMX-Anwendung?

Eine CMX-Anwendung ist eine im Netz unter ihrem GLOBALEN NAMEN identifizierbare Einheit. Sie kann zu anderen Anwendungen im Netz Verbindungen aufbauen, wie in Abschnitt 1.1 dargestellt. Beispiele dafür sind:

- CMX-Anwendungen in SINIX-Rechnern
- DCAM-Anwendungen in BS2000-Rechnern
- PDN-Anwendungen in Datenstationsrechnern

Eine CMX-Anwendung entsteht, wenn sie sich bei CMX anmeldet (Aufruf **t_attach()** oder **x_attach()**).Dabei muß sie ihren LOKALEN NAMEN an CMX übergeben. Der LOKALE NAME ist eine Eigenschaft zum GLOBALEN NAMEN der CMX-Anwendung. Unter dem GLOBALEN NAMEN ist sie im TS-Directory erfaßt.



Anwendung: GLOBALER NAME
Eigenschaft: LOKALER NAME

Bild 1-7 CMX-Anwendung

Jede Anwendung muß die in ihrem System vorgegebenen Konventionen, zum Beispiel zum Verbindungsaufbau, beachten, entsprechend der jeweils gültigen Kommunikationsmethode.

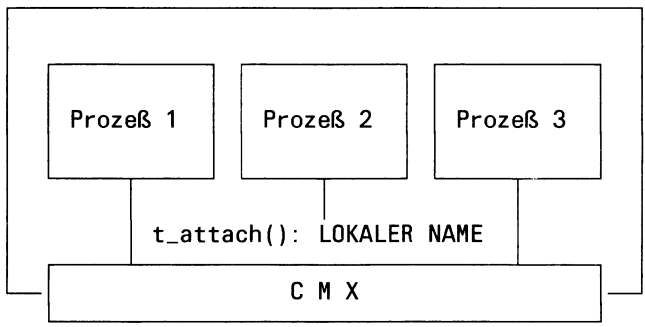
Die Partner müssen sich auch über die Form der zu übertragenden Daten verständigen. Dabei muß beachtet werden, welcher Zeichensatz im jeweiligen System vorliegt. In SINIX-Rechnern ist das der ISO-7-Bit Code, in BS2000- und PDN-Systemen der EBCDIC-Code. Erforderliche Umcodierungen der Daten müssen die Anwendungen selbst vornehmen, da die Übertragung durch die Transportsysteme und CMX codetransparent ist.

Die beiden folgenden Abschnitte beschreiben die Zusammenhänge Anwendung - Prozesse und Prozesse - Verbindungen. Sie gelten in der Regel nur für Rechnerkopplung an WAN oder LAN. Bei Stationskopplung gibt es Einschränkungen, die in Kapitel 8.3 beschrieben sind.

1.3.2 CMX-Anwendungen und SINIX-Prozesse

Im einfachsten Fall ist eine CMX-Anwendung ein einziger SINIX-Prozeß. Zur Strukturierung einer Anwendung gibt es aber weitere Möglichkeiten.

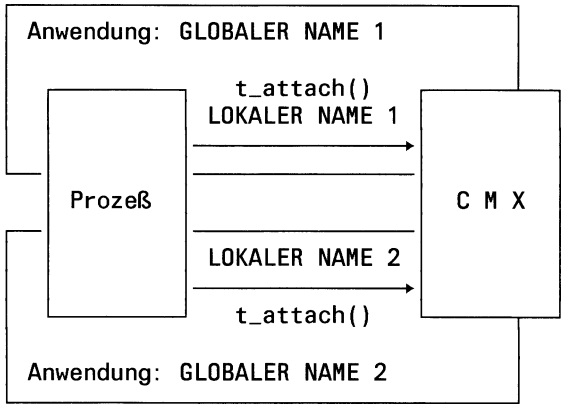
Eine Anwendung kann mit mehreren Prozessen arbeiten. Prozesse gehören zur selben Anwendung, wenn sie sich mit demselben LOKALEN NAMEN bei CMX angemeldet haben. Der erste Prozeß, der sich anmeldet, erzeugt die Anwendung.




```
Anwendung: GLOBALER NAME
Eigenschaft: LOKALER NAME
Funktion: t_attach()
```

Bild 1-8 Eine Anwendung - mehrere Prozesse

Andererseits kann ein Prozeß mehrere Anwendungen steuern. Dazu melden Sie den Prozeß mit verschiedenen LOKALEN NAMEN bei CMX an.




```
Anwendung: GLOBALER NAME 1/2
Eigenschaft: LOKALER NAME 1/2
Funktion: t_attach()
```

Bild 1-9 Ein Prozeß - mehrere Anwendungen

Der Prozeß unterscheidet die verschiedenen Anwendungen, die er steuert, durch die verschiedenen LOKALEN NAMEN.

1.3.3 Verbindungen und SINIX-Prozesse

Ein Prozeß einer CMX-Anwendung kann mehrere Verbindungen halten. Jede Verbindung erhält von CMX eine Identifikation, die Transportreferenz. Damit allein können Sie eine Verbindung gezielt ansprechen. Aber jede Verbindung ist zu jedem Zeitpunkt **genau** einem Prozeß zugegeordnet. Sie ist **nicht** "vererbbar" durch **fork()** [CES].

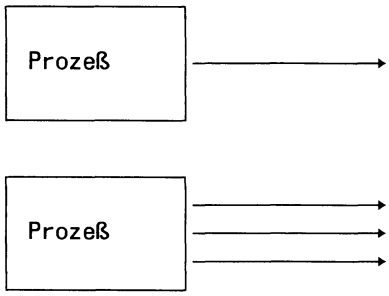


Bild 1-10 Verbindungen und Prozesse

Ein Prozeß kann aber eine Verbindung zu einem anderen Prozeß, der sich in derselben Anwendung angemeldet hat, umlenken. Damit kann man Verbindungen zu verschiedenen Partnern in verschiedenen Prozessen behandeln. Ein zentraler Verteilprozeß kann z.B. alle Verbindungen entgegennehmen und dann an geeignete nachgeordnete Prozesse umlenken.

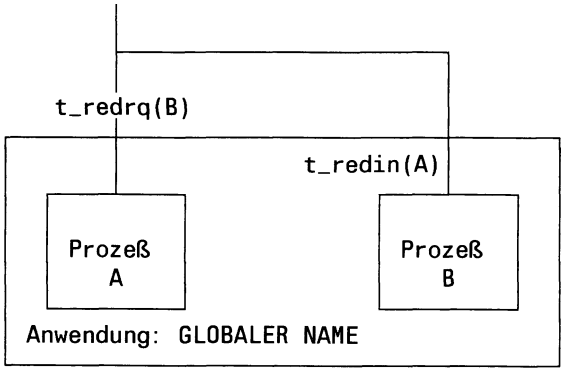


Bild 1-11 Verbindungen umlenken

Die Verbindung ist dann dem Prozeß, der sie abgegeben hat, nicht mehr bekannt.

2 Wie schreibt man eine CMX-Anwendung?

Eine CMX-Anwendung ist ein C-Programm, das die CMX-Funktionen aufruft. Jede CMX-Anwendung hat die folgende Struktur:

```
#include <cmx.h>
#include <tnsx.h>
 .
 .
 .
/*
 * Parameterstrukturen für ICMX(TNS)
 */
 .
 .
 .
main(argc, argv)
int argc;
char *argv[];
{
        .
        .
        .
        ts13_read_properties(); /* Ermittlung LOKALER NAME */
        t_attach();             /* Anmelden bei CMX */
        ts13_read_properties(); /* Ermitteln TRANSPORTADRESSE */
                                /* des Partners */
        t_conrq();              /* Verbindung aufbauen */
        .
        .
        .
        t_concf();              /* Uebernahme Verb.bestaetigung */
        ts11_read_leafs();      /* Ermitteln GLOBALEN NAMEN */
                                /* des Partners */
        .
        .
        t_disrq();              /* Verbindung abbauen */
        t_detach();             /* Abmelden bei CMX */
        .
        .
        exit();
}
```

Bild 2-1 Struktur eines CMX-Anwendungsprogrammes

Die Dateien cmx.h und tnsx.h sind immer als Include-Dateien für CMX nötig. Sie enthalten Definitionen von CMX-Parametern für die Funktionen von ICMX(L) und ICMX(TNS).

Soll die CMX-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so sind im obigen Beispiel folgende Änderungen erforderlich:

- zusätzliche Include-Datei neabx.h
- Der Prefix t_ ist bei allen Anrufen durch x_ zu ersetzen.

Parameterübergabe und Speicherbereitstellung

Parameter zu den CMX-Funktionen werden als Werte oder Zeiger übergeben, für Optionen sind Varianten (union ...) definiert. Alle Strukturen sind in den Include-Dateien definiert.

Achtung

Grundsätzlich müssen Sie alle Speicherbereiche, in denen Sie Werte an CMX übergeben, oder in die CMX etwas eintragen soll, in Ihrem Programm zur Verfügung stellen. Sie belegen solche Speicherbereiche entweder zur Compilierzeit (statisch) oder zur Laufzeit (dynamisch), etwa mit **malloc()** [CES]. In den Parameterstrukturen von CMX sind für Bereiche variabler Länge Längenfelder definiert. In diesen tragen Sie vor dem Aufruf von CMX die Länge des bereitgestellten Bereiches ein. Bei der Rückkehr können Sie daraus dann in der Regel die Länge der von CMX eingetragenen Daten ablesen.

Fehlerabfrage

Ein fehlerhaft abgelaufener Funktionsaufruf kehrt immer mit einer globalen Fehleranzeige zurück. Einen detaillierteren Wert erhalten Sie durch den Aufruf der Fehlerabfragefunktion. Es gilt:

Funktions-aufrufe	globaler Fehlercode	Fehlerabfrage-funktion
t_ ... ts_ ... x_ ...	T_ERROR TS_ERROR X_ERROR	t_error() 1) x_error()

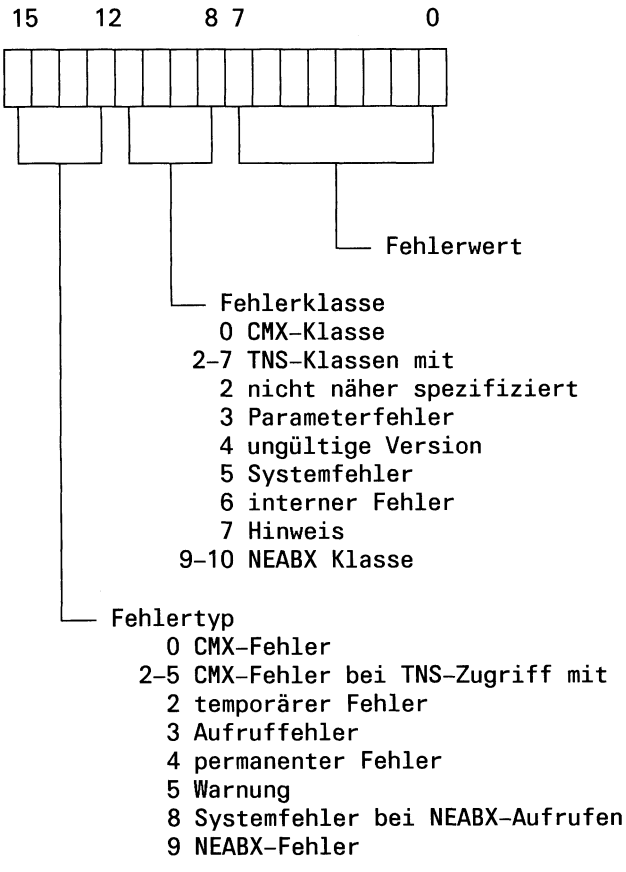
1) der genaue Fehlercode wird von der Funktion selbst im Standardkopf geliefert

Beispiel

```
if (t_attach(...) == T_ERROR) {
    error=t_error();
    fprintf(stderr,"CMX-MELDUNG,Typ %d,Klasse%d,Wert%d\n",
          error>>12&0x0f,error>>8&0x0f,error&0xff);
    exit(1);
}

if (x_attach(...) == X_ERROR) {
    error=x_error();
    fprintf(stderr,"NEABX-Meldung,Typ%d,Klasse%d,Wert%d\n",
          error>>12&0x0f,error>>8&0x0f,error&0xff);
    exit(1);
}

Ts_head ts_head = { TS1_VER01 };
Ts_P13  ts_13 = {
    &ts_head, ...
};

if (ts13_read_properties(&ts13) == TS_ERROR) {
    fprintf(stderr,"TS-MELDUNG, Typ %d, Klasse %d, Wert %d\n",
        ts_head.ts_retcode, ts_head.ts_errclass,
            ts_head.ts_errvalue);
    exit(1);
}
```

Die Darstellung des von
t_error()
oder
x_error()
gelieferten Wertes sollte für Diagnosezwecke wie in den obigen Beispielen erfolgen, da die Fehlerwerte hexadezimal wie folgt codiert sind:

```
Die Fehlermeldung wird in 16-Bit dargestellt.
Nicht aufgeführte Werte sind reserviert.
```



Eine ausführliche Beschreibung der Meldungen ist in Abschnitt 6 bei der Beschreibung der Diagnoseaufrufe t_error(), x_error() sowie im einleitenden Abschnitt zu den Aufrufen der Schnittstelle ICMX(TNS) enthalten.

2.1 Eigenschaften einer CMX-Anwendung

Jede Kommunikationsanwendung besitzt einen GLOBALEN NAMEN. Er gibt in für menschliche Benutzer geeigneter Form an, wie die Anwendung heißt. Die GLOBALEN NAMEN der Kommunikationsanwendung stehen im TS-Directory. Dort sind zum GLOBALEN NAMEN auch Eigenschaften der Kommunikationsanwendung gespeichert.

Zu diesem Eintrag können für eine **lokale** Anwendung die Eigenschaften LOKALER NAME, MIGRATIONSSERVICE und GERAETETYP existieren. Diese Eigenschaften können Sie mit dem Aufruf **ts13_read_properties()** von CMX abfragen.

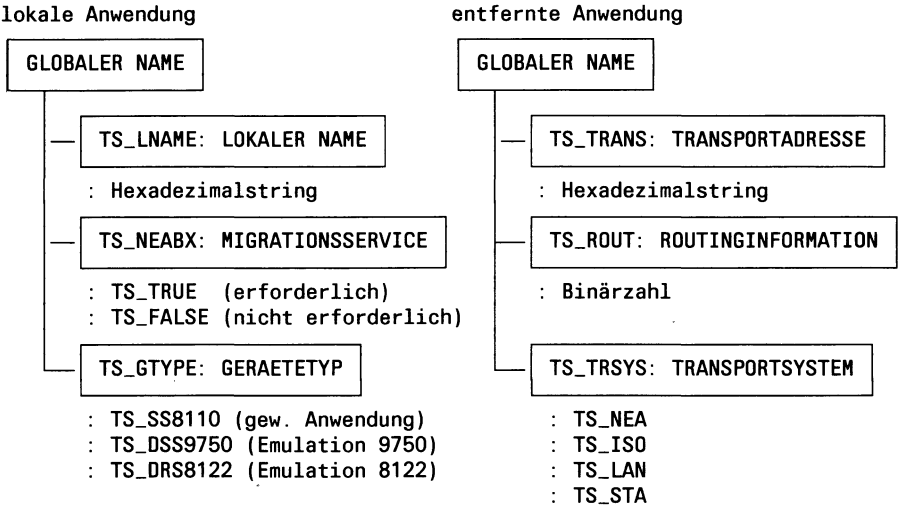


Bild 2-2 Eigenschaften von Anwendungen. Die TS_... sind in tnsx.h definiert

Die Eigenschaft LOKALER NAME einer CMX-Anwendung brauchen Sie, um einen Prozeß für diese Anwendung bei CMX anzumelden (siehe nächster Abschnitt). Der LOKALE NAME ist ein Hexadezimalstring mit nichtabdruckbaren Zeichen.

Aus der Eigenschaft MIGRATIONSSERVICE ermitteln Sie, ob Sie die Aufrufe von ICMX(L),
t_...
oder von ICMX(NEA),
x_...
verwenden müssen.
Diese Eigenschaft hat die möglichen Werte TS_TRUE und TS_FALSE.
Bei TS_FALSE müssen Sie die Aufrufe
t_...
von ICMX(L) programmieren, im anderen Fall die Aufrufe
x_...
von ICMX(NEA).

Achtung

Sie dürfen in einem CMX-Anwendungsprogramm in einer Kommunikationsphase die Aufrufe nicht mischen!

Die Eigenschaft GERAETETYP beschreibt, ob es sich um eine Emulationsanwendung handelt, und wenn ja, um welchen Gerätetyp. Sie brauchen sich mit dieser Eigenschaft nicht zu beschäftigen, wenn Sie eine CMX-Anwendung schreiben. CMX verwendet sie intern für die Migration.

Für eine **entfernte** Anwendung, d.h. eine, die in einem anderen System residiert, sind die Eigenschaften TRANSPORTADRESSE, ROUTINGINFORMATION und TRANSPORTSYSTEM erfaßt. Auch diese Eigenschaften können Sie mit dem Aufruf **ts13_read_properties()** von CMX abfragen.

Die Eigenschaft TRANSPORTADRESSE besitzt als Wert die von CMX erwartete Adresse des Kommunikationspartners beim Verbindungsaufbau. Mit dem Aufruf **ts11_read_leafs()** an CMX können Sie die TRANSPORTADRESSE in den GLOBALEN NAMEN des Kommunikationspartners zurückübersetzen. Die TRANSPORTADRESSE ist ein Hexadezimalstring mit nichtabdruckbaren Zeichen.

Die Eigenschaft ROUTINGINFORMATION beschreibt, über welche von mehreren vorhandenen, gleichen Ankopplungsarten des PC die Kommunikation erfolgen soll (siehe Abschnitt 1.1). Wenn Sie eine CMX-Anwendung schreiben, brauchen Sie sich mit dieser Eigenschaft nicht zu beschäftigen. CMX verwendet sie intern.

Die Eigenschaft TRANSPORTSYSTEM enthält als Wert den Typ des Transportsystems für die Kommunikation zu einem entfernten Kommunikationspartner (siehe Abschnitt 1.1). Wenn Sie eine CMX-Anwendung schreiben, brauchen Sie sich mit dieser Eigenschaft nicht zu beschäftigen. CMX verwendet sie intern für den NEABX Migrationsservice.

Aus der aufgezeigten Struktur der Eigenschaften sehen Sie, daß Sie im Netz einen Kommunikationspartner über genau eine der Ankopplungsarten des PC (Stationskopplung STA oder Rechnerkopplung über WAN oder LAN, siehe Abschnitt 1.1) erreichen können. Aus dem Netz kann eine CMX-Anwendung von verschiedenen Kommunikationspartnern über verschiedene Rechnerkopplungsarten erreicht werden. Außerdem sehen Sie, daß ein Anwendungsprogramm, wenn es in mehreren Rechnern abläuft, jeweils zu einer anderen CMX-Anwendung gehört. Das bedeutet, es wird unter einem anderen GLOBALEN NAMEN laufen. Der Programmname und der GLOBALE NAME der CMX-Anwendung dürfen nicht verwechselt werden. Eine CMX-Anwendung hat einen im Netz eindeutigen GLOBALEN NAMEN. Er wird gemäß den Erfordernissen durch den Netzverwalter vergeben. Dennoch kann sie funktionell aus den selben Programmen in den verschiedenen Endsystemen bestehen.

2.2 Anmelden und Abmelden bei CMX

Jeder Prozeß, der in einer CMX-Anwendung arbeiten will, muß sich für diese CMX-Anwendung bei CMX anmelden. Das erfolgt mit dem Aufruf **t_attach().** Sie übergeben dabei den LOKALEN NAMEN der CMX-Anwendung und verschiedene Parameter, die das Verhalten des Prozesses in dieser Anwendung steuern. Durch die Anmeldung des ersten Prozesses entsteht die Anwendung für CMX. CMX beginnt nun, für diese Anwendung Verbindungswünsche entgegenzunehmen. Erst, wenn der Prozeß angemeldet ist, können Sie weitere CMX-Funktionen **t_...** aufrufen. Bevor Sie einen Prozeß beenden, rufen Sie **t_detach()** auf, um den Prozeß bei CMX für diese Anwendung abzumelden. Zuvor müssen Sie aber alle Verbindungen (siehe Abschnitt 3) abbauen. Tun Sie das nicht, baut CMX implizit selbst alle Verbindungen ab. Dies ist aber nur für Ausnahmesituationen, etwa, wenn sich ein Prozeß unerwartet frühzeitig beendet, vorgesehen.

Sobald sich der letzte Prozeß aus einer Anwendung abgemeldet hat, existiert die Anwendung für CMX nicht mehr. Verbindungswünsche von Kommunikationspartnern werden für diese Anwendung nicht mehr angenommen.

Soll die CMX-Anwendung über die Migrationsschnittstelle CMX(NEA) kommunizieren, so sind anstatt t_attach(), t_detach und t_... die Aufrufe x_attach(), x_detach und x_ ... zu verwenden.

2.3 Zur Kommunikation gehören zwei

Nachdem sich ein Prozeß bei CMX angemeldet hat, gehört er zu einer lokalen Anwendung. Ein Partner ist immer eine andere Anwendung, die in einem anderen Rechner abläuft.

Der Prozeß der lokalen Anwendung kann aktiv einen Kommunikationspartner ansprechen, das heißt, er kann aktiv eine Verbindung aufbauen (siehe Abschnitt 3.1). Er repräsentiert dann die "rufende Anwendung". Er kann aber auch warten, bis ein Kommunikationspartner sich mit einem Verbindungswunsch an ihn wendet. Das heißt dann, er möchte passiv eine Verbindung entgegenehmen. Er repräsentiert dann die "gerufene Anwendung".

In beiden Fällen kommt er in einen Zustand, wo er auf eine Antwort vom Kommunikationspartner oder von CMX wartet.

2.4 Ereignisse

CMX meldet dem Prozeß alle Ereignisse, die er empfängt. Solche Ereignisse können z.B. sein:

- der Verbindungswunsch eines Kommunikationspartners (der "rufenden Anwendung")
- die Ankunft von Daten auf einer bestehenden Verbindung
- der Verbindungsabbau durch den Kommunikationspartner oder CMX

Jedes Ereignis müssen Sie mit einem Aufruf **t_event()** abfragen.

Dabei haben Sie zwei Möglichkeiten:

1. Synchrone Verarbeitung

Sie rufen **t_event()** mit Parameter cmode = T_WAIT auf. Solange kein Ereignis ansteht, schläft der Prozeß und verbraucht keine Rechenzeit. Bei einem Ereignis weckt CMX den Prozeß auf und **t_event()** bringt als Ergebnis den Code des Ereignisses sowie ggfs. die Transportreferenz der betroffenen Verbindung.

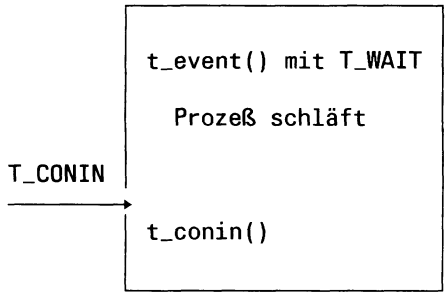


Bild 2-3 Synchrone Verarbeitung

Auch wenn der Prozeß in **t_event()** schläft, kann man ihn mit Signalen [CES] aufwecken. CMX setzt ihn dann mit T_NOEVENT fort, falls eine Behandlung für dieses Signal definiert ist.

2. Asynchrone Verarbeitung

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

Es ist aber nicht sinnvoll, nur **t_event()** in einer Schleife laufen zu lassen, besser sollten Sie dann cmode = T_WAIT verwenden, und den Prozeß, wenn nötig, periodisch mit **alarm()** [CES] aufwecken.

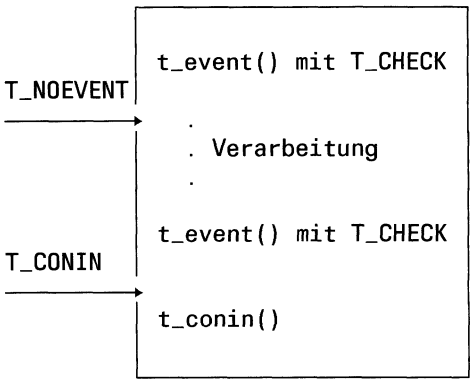


Bild 2-4 Asynchrone Verarbeitung

Abhängig davon, welches Ereignis gemeldet wurde, erwartet CMX 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). Soll die CMX-Anwendung über die Migrationsschnittstelle CMX(NEA) kommunizieren, so müssen Sie die Ereignisse mit dem Aufruf x_event() abfragen.

2.5 Zustandsautomaten

Die Abläufe zur Nutzung der Schnittstelle ICMX(L) können mit Zustandsautomaten dargestellt werden. Das sind Diagramme, die die Zustände einer CMX-Anwendung und die Zustandsübergänge enthalten. Aus diesen Diagrammen kann man ablesen, welche Aufrufe nur in bestimmten Zuständen erlaubt sind.

Die Abläufe können in vier Phasen aufgeteilt werden. Die Phasen bilden eine hierarchische Struktur, wobei die Phase A der höchsten und die Phase D der niedrigsten Hierarchiestufe entspricht.

Jede Phase wird durch einen oder mehrere Automaten dargestellt. Die doppelt gepunkteten Rechtecke eines Automaten stellen den Zustand dar, in dem die Automaten der nächsten Phase bzw. niedrigeren Hierarchiestufe aktiviert wurden. Der Übergang aus einem Zustand, der durch ein doppelt gepunktetes Rechteck dargestellt ist, in einen anderen Zustand bewirkt, daß die zugeordneten Automaten der niedrigeren Hierarchiestufe inaktiviert werden.

Phase A dient der Aktivierung einer CMX-Anwendung, Phase B der Anmeldung, Phase C dem Verbindungsauf- und -abbau, Phase D dem Datenaustausch.

Die einzelnen Zustände der Automaten selbst und die Zustandsübergänge sind in Kapitel 6.1 für ICMX(L) und Kapitel 6.2 für ICMX(NEA) beschrieben.

3 Verbindungen zu Kommunikationspartnern

Welche Ankopplungsarten Ihr PC bietet, haben Sie bereits in Abschnitt 1.1 gelesen. Nun geht es darum, wie man eine Verbindung aufbaut, weitergibt und wieder abbaut.

Verbindungsaufbau und -abbau laufen zwischen zwei Kommunikationspartnern ab. Der eine ist der rufende Partner, er initiiert den Verbindungsaufbau. Der andere ist der gerufene Partner, mit dem der rufende eine Verbindung eingehen will. Die folgenden Abschnitte verdeutlichen die Zusammenhänge und Abläufe.

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

3.1 Verbindung aufbauen

Die Aufrufe in beiden Programmen laufen zeitlich so ab:

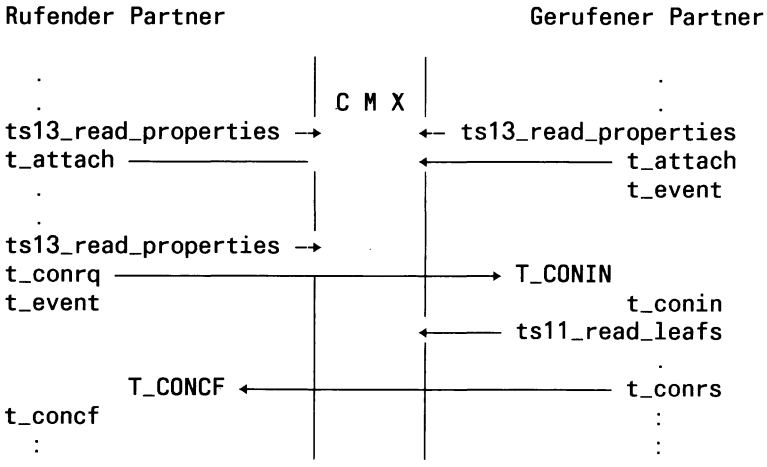


Bild 3-1 Verbindung aufbauen

Was tut der rufende Partner?

Der rufende Partner holt sich zuerst seinen LOKALEN NAMEN und meldet sich dann bei CMX an. Danach ermittelt er die TRANSPORT-ADRESSE des gerufenen Partners und fordert mit **t_conrq()** einen Verbindungsaufbau an. Dann wartet er mit t_event() darauf, daß die Bestätigung des gerufenen Partners, das heißt, das Ereignis T_CONCF eintrifft. Wenn t_event() das Ereignis gemeldet hat, stellt der aktive Partner mit dem Aufruf **t_concf()** die Verbindung her.

Was tut der gerufene Partner?

Der gerufene Partner wartet nach der Anmeldung zunächst mit t_event() auf ein Ereignis. Das Ereignis T_CONIN zeigt den Verbindungsaufbau des rufenden Partners an. Mit dem Aufruf **t_conin()** nimmt der gerufene Partner ihn an. Er ermittelt dann aus der TRANSPORTADRESSE des rufenden Partners dessen GLOBALEN NAMEN und beantwortet den Verbindungswunsch mit **t_conrs().**

Verbindungswunsch ablehnen

Der gerufene Partner 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).

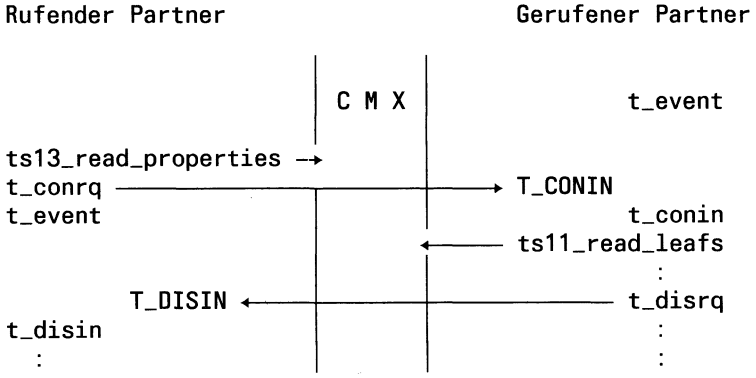


Bild 3-2 Verbindungswunsch ablehnen

Benutzerdaten austauschen beim Verbindungsaufbau

Der Grund, warum die Aufrufe **t_conin()** und **t_concf()** nötig sind, ist, daß beide Partner schon beim Verbindungsaufbau Benutzerdaten und Optionen austauschen können, wenn das Transportsystem solche bietet (siehe Abschnitt 1.2.4)

Bei **t_conrq()** kann der rufende Partner Benutzerdaten mitgeben: eine kleine Datenmenge, die der gerufene Partner bei **t_conin()** erhält. Wenn der gerufene Partner dann den Verbindungswunsch mit **t_conrs()** beantwortet, kann er wiederum Informationen mitgeben. Diese erhält der rufende Partner bei **t_concf()**.

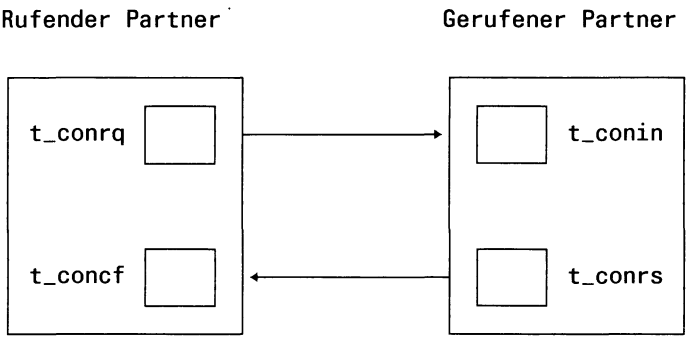


Bild 3-3 Benutzerdatenaustausch beim Verbindungsaufbau

Vorrangdaten aushandeln

Wenn das Transportsystem die Option Vorrangdaten bietet, können die Partner beim Verbindungsaufbau deren Verwendung aushandeln. Das geht wie folgt:

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

Wenn einer der Partner mit dem Ergebnis der Vorrangdatenverhandlung nicht einverstanden ist, kann er die Verbindung abbauen.

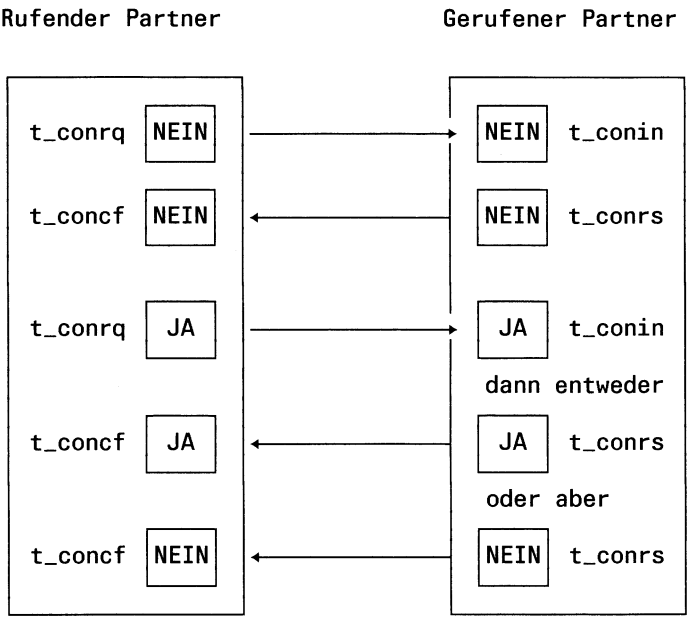


Bild 3-4 Vorrangdatenverhandlung beim Verbindungsaufbau

Soll die CMX-Anwendung über die Migrationsschnittstelle CMX(NEA) kommunizieren, so müssen Sie für die o.g. Aufrufe den Prefix x_ anstatt t_ und für die Ereignisse den Prefix X_ anstatt T_ verwenden.

3.2 Verbindung abbauen

Jeder der beiden Partner kann **t_disrq()** aufrufen, um die Verbindung abzubauen. Der andere Partner erhält das Ereignis T_DISIN. Mit dem Aufruf **t_disin()** nimmt er den Verbindungsabbau entgegen. Dabei erfährt er den Grund für den Verbindungsabbau.

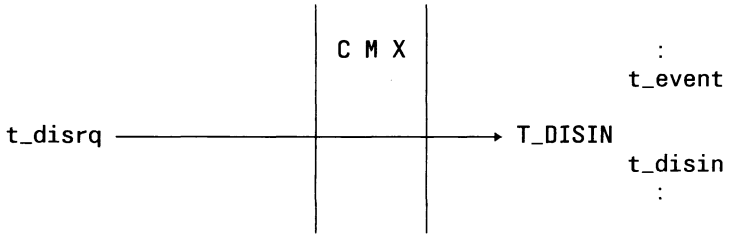


Bild 3-5 Verbindung abbauen

Wenn das Transportsystem die entsprechende Option bietet, kann der Partner, der die Verbindung abbaut, bei **t_disrq()** Benutzerdaten mitschicken. Der andere Partner erhält sie bei **t_disin().**

Auch CMX kann die Verbindung abbauen. Beide Partner erhalten dann das Ereignis T_DISIN, das sie mit **t_disin()** abholen müssen. Aus dem Verbindungsabbaugrund kann jeder Partner ermitteln, ob der andere oder CMX die Verbindung abgebaut hat.

Soll die CMX-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen Sie anstatt t_disrq() und t_disin() die Aufrufe x_disrq() und x_disin() verwenden.

3.3 Beispiel zum Verbindungsauf- und -abbau mit ICMX(L)

Die beiden folgenden Programme zeigen, wie man eine Verbindung aufbaut.
Das Programm conn_a.c repräsentiert die rufende Anwendung,
das Programm conn_p.c repräsentiert die gerufene Anwendung.

Beispiele zum Verbindungsauf- und -abbau mit ICMX(NEA) finden Sie bei der Beschreibung der entsprechenden Aufrufe im Abschnitt 6.3.

```
/*
 * CMX Test Command Source File
 *
 * Aktiver Partner fuer Verbindungsaufbautest:
 * Der aktiver Partner baut die Verbindung zum passiven Partner zyklisch
 * auf und ab.
 * Die Verarbeitung ist TS-Ereignis-gesteuert, es wird synchron gewartet.
 * Im Fehlerfall erfolgt t_detach() (mit implizitem Verbindungsabbau).
 *
 * Alle Ausgaben erfolgen auf stderr, Fehlermeldungen unbedingt, andere
 * Meldungen abhaengig von -d Option
 *
 * Aufruf:
 *      conn_a [-d] [-f] [-tsecs] [-ncount] [-omyname] [-ppartner] [-r]
 *              d       debugging
 *              f       stderr in Datei "ERRxxxxx" mit pid xxxxx umleiten
 *              n       count Verbindungszyklen (default: 1000);
 *              o       eigener Name myname
 *                      (default: "TestanwendungAKT")
 *              p       Partnername partner
 *                      (default: "TestanwendungPAS")
 *              r       nur t_attach in Schleife
 *              t       secs Sekunden (default: 2) zwischen zwei Zyklen warten
 *              u       Benutzerdaten verwenden
 */
static char sccsid[] = "@(#)conn_a.c    1.26 87/01/30 (DF154)";

#include        <stdio.h>
#include        <signal.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <sys/types.h>
#include        <sys/timeb.h>

#define TRUE    1
#define FALSE   0

#define MAXCNT  1000
int     count = MAXCNT;
int     currcnt = 0;

#define MAXSEC  2
int     secs = MAXSEC;

int     tref;                   /* Transportreferenz */
int     reason;                 /* Grund fuer Verbindungsabbau */
int     pid;                    /* eigene Prozessid */
int     ratt = 0;               /* Flag fuer An- Abmeldung */
int     sig;                    /* Merker fuer Signale */
char    rest[9];                /* Enthaelt die absolute Rechnerzeit
                                   in <min>:<sek>.<millisek> */

/*
 * Strukturen fuer Adressierung
 */
#define PNAME   "TestanwendungPAS"
#define MYNAME  "TestanwendungAKT"

char myname[TS_LPN+1] = { MYNAME } ;
char pname[TS_LPN+1] = { PNAME } ;
union   t_address myaddress;
union   t_address partaddress;

struct  t_opta1 t_opta1 = { T_OPTA1, T_ACTIVE, 1 };
```

```
char    udatbuf[T_MSG_SIZE];
struct  t_optc1 t_optc1 = { T_OPTC1, udatbuf, T_MSG_SIZE, T_NO, T_NO };

int     debug = 0;
extern  int     errno;
int     interrupt();            /* Zum Fangen von SIGINT (DEL) */

        /* PARAMETERSTRUKTUREN FUER DIRECTORY SYSTEM */
        /* ========================================= */

              /* Standardkopf */
Ts_head       ts_head = { TS_1VER01 } ;

        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Properties" */
        /* ------------------------------------------------------ */

Ts_leaf_name ts_myname   = { 1, TS_PN, NULL, myname } ;
Ts_leaf_name ts_partname = { 1, TS_PN, NULL, pname } ;

Ts_property ts_prop[] = { NULL, TS_ITEM, NULL, NULL,
                          TS_EMPTYPROP };

Ts_P13 ts1003 = { &ts_head,             /* Standardkopf                    */
                  TS_CMX_DIR_ID,        /* Directory-Identifier            */
                  NULL,                 /* Objektname                      */
                  NULL,                 /* irrelevanter Parameter          */
                  NULL,                 /* Werteliste fuer Eigenschaften   */
                                        /* Laenge der Werteliste           */
                  sizeof (union t_address),
                  ts_prop,              /* Liste der gewuenschten Eigensch. */
                  NULL, NULL, NULL } ; /* irrelevanter Parameter           */

main(argc, argv)
int argc;
char *argv[];
{
        register        i;
        int     mode;
        char    errfil[20];
        int     error = 0;
        int     file = 0;
        int     udat = 0;
        int     event;

        /*
         * Optionsanalyse
         */
        while (--argc && **++argv == '-') {
                switch (*++*argv) {
                case 'd':
                        do {
                                debug++;
                        } while (*++*argv) ;
                        break;
                case 'f':
                        file++;
                        break;
                case 'n':
                        count = atoi(++*argv);
                        break;
                case 'o':
                        strcpy(myname, ++*argv);
                        break;
                case 'p':
```

```
                strcpy(pname, ++*argv);
                break;
        case 'r':
                ratt++;
                break;
        case 't':
                secs = atoi(++*argv);
                break;
        case 'u':
                udat++;
                break;
        }
}
/*
 * Zur Beendigung des Programms wird SIGINT verwendet,
 * in der Behandlung wird die Anwendung abgemeldet (implizite
 * Verbindungsabbauten). SIGINT soll auch im Hintergrund wirken,
 * daher ignorieren wir die vorherige Signalbehandlung
 */
signal(SIGINT, interrupt);
/*
 * Eventuell stderr ummmelden, ungepufferte Ausgabe
 */
pid = getpid();
if (file) {
        sprintf(errfil, "ERR%05d", pid);
        freopen(errfil, "w", stderr);
        setbuf(stderr, NULL);
}
fprintf(stderr, "count %d, secs %d, myname '%s', pname '%s'\n",
                count, secs, myname, pname);

/*
 * Aktive Anwendung beim Driver anmelden
 */
ts1003.ts13leaf_name = &ts_myname ;
ts_myname.ts_lf[0].ts_np_length = strlen (myname) ;
ts1003.ts13prval = &myaddress ;
ts_prop[0].ts_name = TS_LNAME ;
if (ts13_read_properties (&ts1003) == TS_ERROR) {
        fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                ts_head.ts_retcode, ts_head.ts_errclass,
                ts_head.ts_errvalue) ;
        exit (FALSE) ;
} else {
        if (ts_prop[0].ts_length == 0) {
                fprintf (stderr, "TS-Fehler
                (Eigenschaft 'eigener Name' nicht gefunden)\n") ;
                exit (FALSE) ;
        }
}

if (t_attach(&myaddress, &t_opta1) == T_ERROR) {
        fprintf(stderr, ">>> FEHLER %d bei t_attach\n", t_error());
        exit(FALSE);
}
fprintf(stderr, "%02d:%s   Anwendung '%s' angemeldet, pid %d\n",
        rechzeit(rest), rest, myname, pid);

/*
 * Verbindung aufbauen zum passiven Partner
 */
ts1003.ts13leaf_name = &ts_partname ;
ts_partname.ts_lf[0].ts_np_length = strlen (pname) ;
```

```
ts1003.ts13prval = &partaddress ;
ts_prop[0].ts_name = TS_TRANS ;
if (ts13_read_properties (&ts1003) == TS_ERROR) {
        fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                ts_head.ts_retcode, ts_head.ts_errclass,
                ts_head.ts_errvalue) ;
        exit (FALSE) ;
} else {
        if (ts_prop[0].ts_length == 0) {
                fprintf (stderr, "TS-Fehler
                (Eigenschaft 'Partneradresse' nicht gefunden)\n") ;
                exit (FALSE) ;
        }
}
sprintf(udatbuf, "myaddr: '%s', paddr: '%s'", myname, pname);
t_optc1.t_udatal = (udat ? strlen(udatbuf) : 0);
if (t_conrq(&tref, &partaddress, &myaddress, &t_optc1) == T_ERROR) {
        fprintf(stderr, ">>> FEHLER %d bei t_conrq, tref 0x%x\n",
                t_error(), tref);
        detach(FALSE);
}

/*
 * Ereignisgesteuerte Verarbeitung:
 * i.a. t_event() synchron (T_WAIT) wartend
 */
mode = T_WAIT;
for (;;) {
        switch (event = t_event(&tref, mode, NULL)) {

        case T_CONCF:
                /*
                 * Verbindungsaufbau gelungen?
                 */
                t_optc1.t_udatal = T_MSG_SIZE;
                if (t_concf(&tref, (udat ? &t_optc1: NULL))
                   == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d bei
                        t_concf tref 0x%x\n",
                                t_error(), tref);
                        detach(FALSE);
                }
                if (udat && t_optc1.t_udatal)
                        fprintf(stderr,"\t%s\n", udatbuf);
                currcnt++;
                if (debug) {
                        fprintf(stderr, "%02d:%s   %3d. von
                        %d Verbindungen zu '%s' aufgebaut\n",
                                rechzeit(rest), rest, currcnt,
                                count, pname);
                } else
                        fprintf(stderr, ".");
                /*
                 * Verbindung abbauen und neu aufbauen falls
                   bis hier OK
                 */
                if (t_disrq(&tref, NULL) == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d
                        bei t_disrq tref 0x%x\n",
                                t_error(), tref);
                        detach(FALSE);
                }
                /*
                 * Bei Zaehlerablauf abmelden
```

```
                        */
                        if (currcnt == count)
                                detach(TRUE);
                        /*
                         * Verbindung wieder aufbauen zum passiven Partner
                         */
                        if (secs)
                                sleep(secs);
                        if (udat) {
                                sprintf(udatbuf, "myaddr: '%s',
                                paddr: '%s'", myname, pname);
                                t_optc1.t_udatal = strlen(udatbuf) + 1;
                        } else
                                t_optc1.t_udatal = 0;
                        if (t_conrq(&tref, &partaddress, &myaddress,
                           &t_optc1) == T_ERROR) {
                                fprintf(stderr, ">>> FEHLER %d
                                bei t_conrq fuer tref 0x%x\n",
                                        t_error(), tref);
                                detach(FALSE);
                        }
                        break;

                case T_ERROR:
                        fprintf(stderr, ">>> FEHLER %d bei t_event\n",
                                t_error());
                        detach(FALSE);
                        break;

                case T_DISIN:
                        /*
                         * Verbindungsabbau durch System
                         */
                        if (t_disin(&tref, &reason, NULL) == T_ERROR) {
                                fprintf(stderr, ">>> FEHLER %d
                                bei t_disin tref 0x%x\n",
                                        t_error(), tref);
                                detach(FALSE);
                        }
                        if (debug) {
                                fprintf(stderr, "%02d:%s   Verbindungs-
                                sabbau erhalten, tref 0x%x, reason %d\n",
                                        rechzeit(rest), rest, tref, reason);
                        }
                        detach(FALSE);
                        break;

                case T_NOEVENT:
                        /*
                         * Abbruch durch Signal
                         */
                        if (sig)
                                detach(TRUE);
                        break;

                default:
                        fprintf(stderr, "Unerwarteter event %d\n", event);
                        detach(FALSE);
                }
        }
}

detach(result)
{
```

```
        if (t_detach(&myaddress) == T_ERROR)
                fprintf(stderr, ">>> FEHLER %d bei t_detach\n", t_error());
        fprintf(stderr, "%02d:%s   Anwendung '%s', pid %d, abgemeldet %s\n",
                rechzeit(rest), rest, myname, pid,
                result == FALSE ? "wegen Fehler" : "");
        fflush(stderr);
        fclose(stderr);
        exit(result);
}

/*
 * Signalroutine fuer SIGINT
 */
interrupt()
{
        sig++;
}

/* ============================================================
 * Die Funktion rechnet die absolute Zeit, in der die be-
 * stimmte Tat ausgefuert ist.
 *  EINGABE: 'wert' Pointer zu einem Feld wo die Zeit gespeichert wird.
 *  AUSGABE: 'hass' Stundewert
 */
rechzeit(wert)
char *wert;
{
        struct timeb pt;
        int hass, mass, dec;

        ftime(&pt);
        if((hass = pt.time/3600%24 - pt.timezone/60) == 24)     /* Stunden */
                hass = 0;
        mass = pt.time%3600/60;          /* Minuten */
        dec = pt.time%3600%60;           /* Sekunden */
        sprintf(wert,"%02d:%02d.%03u", mass, dec, pt.millitm);
        return(hass);
}
```

```
/*
 * CMX Test Command Source File
 *
 * Passiver Partner fuer Verbindungsaufbautest
 * Der passive Partner erwartet den Verbindungsaufbau (und -abbau) seitens
 * des aktiven Partners.
 * Der passive Partner baut die Verbindung aus eigener Initiative ueberhaupt
 * nicht ab, sonst erwartet den Abbau durch den aktiven Partner.
 * Die Verarbeitung erfolgt TS-Ereignis-gesteuert.
 * Alle Ausgabe erfolgen auf stderr, Fehlermeldungen unbedingt, andere
 * Meldungen abhaengig von -d Option
 *
 * Aufruf:
 *      conn_p [-d] [-f] [-ncount] [-omyname] [-ppartner] [-rreason] [-u]
 *              d       debugging
 *              f       stderr in Datei "ERRxxxxx" mit pid xxxxx umleiten
 *              n       count Verbindungsaufbauten (default: 1000)
 *              o       eigener Name myname
 *                      (default: "TestanwendungPAS")
 *              p       Partnername partner
 *                      (default: "TestanwendungAKT")
 *              r       Reason als Dezimalzahl (default: T_USER (=0));
 *                      bei Klasse 0 = 256
 *              u       Benutzerdaten verwenden
 */
static char sccsid[] = "@(#)conn_p.c    1.24 87/01/30 (DF154)";

#include        <stdio.h>
#include        <signal.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <sys/types.h>
#include        <sys/timeb.h>

#define TRUE    1
#define FALSE   0

#define MAXCNT  1000
int     count = MAXCNT;
int     currcnt;

int     tref;                   /* Transportreferenz */
int     pid;                    /* eigene pid */
int     reason1;                /* Grund fuer Verbindungsabbau */
int     reason2 = T_USER;       /* Grund fuer Verbindungsabbau */
int     sig;                    /* Merker fuer Signale */
char    rest[9];                /* Entaehlt der Rechnerzeit in
                                   <min>:<sek>.<millisek> */

/*
 * Strukturen fuer Adressierung
 */
#define MYNAME  "TestanwendungPAS"
#define PNAME   "TestanwendungAKT"

char myname[TS_LPN+1] = { MYNAME } ;
char pname[TS_LPN+1] = { PNAME } ;
union   t_address myaddress;
```

```
union   t_address partaddress;

struct  t_opta1 t_opta1 = { T_OPTA1, T_PASSIVE, 1 };
char    udatbuf[T_MSG_SIZE];
struct  t_optc1 t_optc1 = { T_OPTC1, udatbuf, T_MSG_SIZE, T_NO, T_NO };

int     debug = 0;
extern  int     errno;
int     interrupt();            /* Zum Fangen von SIGINT (DEL) */

        /* PARAMETERSTRUKTUREN FUER DIRECTORY SYSTEM */
        /* ========================================= */

              /* Standardkopf */
Ts_head       ts_head = { TS_1VER01 } ;

        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Leafs" */
        /* ------------------------------------------------- */

Ts_leaf_name ts_sname = { 0 } ;
Ts_leaf_name ts_pname ;
char         ts_nlist[TS_LPN] ;
Ts_filter ts_filter = { TS_TRANS, NULL, (char *) &partaddress } ;

Ts_P11 ts1001 = { &ts_head,          /* Standardkopf                  */
                  TS_CMX_DIR_ID,     /* Directory-Identifier          */
                  &ts_sname,         /* Suchkriterium (Name)          */
                  TS_GENERAL,        /* Suchmodus                     */
                  &ts_filter,        /* Suchkriterium (Eigenschaft)   */
                  TS_LOCAL,          /* Suchmodus                     */
                  NULL, NULL,        /* irrelevante Parameter         */
                  ts_nlist,          /* Werteliste fuer Namensteile   */
                  sizeof(ts_nlist), /* Laenge der Werteliste         */
                  1,                 /* Anzahl der erwarteten Namen   */
                  &ts_pname,         /* Liste fuer gefundene Objekte  */
                  NULL } ;           /* Anzahl der gefundenen Objekte */
                                     /* (Rueckgabeparameter)          */

        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Properties" */
        /* ------------------------------------------------------ */

Ts_leaf_name ts_myname   = { 1, TS_PN, NULL, myname } ;

Ts_property ts_prop[] = { NULL, TS_ITEM, NULL, NULL,
                          TS_EMPTYPROP };

Ts_P13 ts1003 = { &ts_head,             /* Standardkopf                    */
                  TS_CMX_DIR_ID,        /* Directory-Identifier            */
                  NULL,                 /* Objektname                      */
                  NULL,                 /* irrelevanter Parameter          */
                  NULL,                 /* Werteliste fuer Eigenschaften   */
                                        /* Laenge der Werteliste           */
                  sizeof (union t_address),
                  ts_prop,              /* Liste der gewuenschten Eigensch. */
                  NULL, NULL, NULL } ; /* irrelevanter Parameter          */

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

```
main(argc, argv)
int argc;
char **argv;
{
        register        i;
        int     mode;
        char    errfil[20];
        int     error = 0;
        int     file = 0;
        int     udat = 0;
        int     event;

        /*
         * Optionsanalyse
         */
        while (--argc && **++argv == '-') {
                switch (*++*argv) {
                case 'd':
                        do {
                                debug++;
                        } while (*++*argv) ;
                        break;
                case 'f':
                        file++;
                        break;
                case 'n':
                        count = atoi(++*argv);
                        break;
                case 'o':
                        strcpy(myname, ++*argv);
                        break;
                case 'p':
                        strcpy(pname, ++*argv);
                        break;
                case 'r':
                        reason2 = atoi(++*argv);
                        break;
                case 'u':
                        udat++;
                        break;
                }
        }
        /*
         * Zur Beendigung des Programms wird SIGINT verwendet,
         * in der Behandlung wird die Anwendung abgemeldet (implizite
         * Verbindungsabbauten). SIGINT soll auch im Hintergrund wirken,
         * daher ignorieren wir die vorherige Signalbehandlung
         */
        signal(SIGINT, interrupt);
        /*
         * Evtl. stderr umlenken, ungepufferte Ausgabe
         */
        pid = getpid();
        if (file) {
                sprintf(errfil, "ERR%05d", pid);
                freopen(errfil, "w", stderr);
                setbuf(stderr, NULL);
        }
```

```
fprintf(stderr, "count %d, myname '%s', pname '%s'\n",
        count, myname, pname);
/*
 * Passive Anwendung beim Driver anmelden
 */
ts1003.ts13leaf_name = &ts_myname ;
ts_myname.ts_lf[0].ts_np_length = strlen (myname) ;
ts1003.ts13prval = &myaddress ;
ts_prop[0].ts_name = TS_LNAME ;
if ( ts13_read_properties( &ts1003 ) == TS_ERROR) {
        fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                ts_head.ts_retcode, ts_head.ts_errclass,
                ts_head.ts_errvalue) ;
        exit (FALSE) ;
} else {
        if (ts_prop[0].ts_length == 0) {
                fprintf (stderr, "TS-Fehler (Eigenschaft
                'eigener Name' nicht gefunden)\n") ;
                exit (FALSE) ;
        }
}
if (t_attach(&myaddress, &t_opta1) == T_ERROR) {
        fprintf(stderr, ">>> FEHLER %d bei t_attach\n", t_error());
        exit(FALSE);
}
fprintf(stderr, "%02d:%s   Anwendung '%s' angemeldet, pid %d\n",
        rechzeit(rest), rest, myname, pid);

/*
 * Ereignisgesteuerte Verarbeitung
 */
mode = T_WAIT;
for (;;) {
        switch (event = t_event(&tref, mode, NULL)) {

        case T_CONIN:
                /*
                 * Verbindungsaufbauwunsch akzeptieren
                 */
                t_optc1.t_udatal = T_MSG_SIZE;
                if (t_conin(&tref, &myaddress, &partaddress,
                        (udat ? &t_optc1 : NULL)) == T_ERROR){
                        fprintf(stderr, ">>> FEHLER %d
                        bei t_conin tref 0x%x\n",
                                t_error(), tref);
                        detach(FALSE);
                }
                if (udat && t_optc1.t_udatal)
                        fprintf(stderr,"\t%s\n", udatbuf);
                if (t_conrs(&tref, NULL) == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d
                        bei t_conrs tref 0x%x\n",
                                t_error(), tref);
                        detach(FALSE);
                }
                currcnt++;
                if (debug) {
                        ts_filter.ts_pr_length = partaddress.
```

```
                        tpartaddr.t_palng ;
                        if (ts11_read_leafs (&ts1001) == TS_ERROR) {
                                fprintf (stderr, "TS-Fehler
                                (Typ: %d Klasse: %d Wert: %d)\n",
                                        ts_head.ts_retcode,
                                        ts_head.ts_errclass,
                                        ts_head.ts_errvalue) ;
                                exit (FALSE) ;

                        }
                        ts_nlist[ts_pname.ts_lf[0].ts_np_
                                 length] = '\0' ;
                        fprintf(stderr, "%02d:%s   %3d. von %d
                        Verbindungen von '%s' akzeptiert\n",
                                rechzeit(rest), rest, currcnt,
                                count, ts_nlist);
                } else
                        fprintf(stderr, ".");
                break;

        case T_ERROR:
                fprintf(stderr, ">>> FEHLER %d bei
                t_event\n", t_error());
                detach(FALSE);
                break;

        case T_NOEVENT:
                /*
                 * Abbruch durch Signal
                 */
                if (sig)
                        detach(TRUE);
                break;

        case T_DISIN:
                /*
                 * Verbindungsabbau durch aktiven Partner bei normalem
                 * normalen Ende, oder bei beim Partner auf-
                 * getretenen Fehlern
                */
                if (t_disin(&tref, &reason1, NULL) == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d
                        bei t_disin tref 0x%x\n",
                                t_error(), tref);
                        error++;
                }
                if (reason1 != reason2) {
                        fprintf(stderr, "%02d:%s   Verbindungs-
                        abbau erhalten, tref 0x%x, reason %d\n",
                                rechzeit(rest), rest, tref, reason1);
                        detach(FALSE);
                }
                if (currcnt == count)
                        detach(TRUE);
                break;

        default:
                fprintf(stderr, "Unerwarteter event\n", event);
```

```
                        detach(FALSE);
                        break;
                }
        }
}

detach(result)
{
        if (t_detach(&myaddress) == T_ERROR)
                fprintf(stderr, ">>> FEHLER %d bei t_detach\n", t_error());
        fprintf(stderr, "%02d:%s   Anwendung '%s', pid %d, abgemeldet\n",
                rechzeit(rest), rest, myname, pid,
                result == FALSE ? "nach Fehlern" : "");
        fflush(stderr);
        fclose(stderr);
        exit(result);
}

/*
 * Signalroutine fuer SIGINT
 */
interrupt()
{
        sig++;
}
/* ========================================================================
 * Die Funktion rechnet die absolute Zeit, in der die
 * bestimmte Tat ausgefuert ist.
 *  EINGABE: 'wert' Pointer zu einem Feld wo die Zeit gespeichert wird.
 *  AUSGABE: 'hass' Stundewert
 */
rechzeit(wert)
char *wert;
{
        struct timeb pt;
        int hass, mass, dec;
        ftime(&pt);
        if((hass = pt.time/3600%24 - pt.timezone/60) == 24)     /* Stunden */
                hass = 0;
        mass = pt.time%3600/60;          /* Minuten */
        dec = pt.time%3600%60;           /* Sekunden */
        sprintf(wert,"%02d:%02d.%03u", mass, dec, pt.millitm);
        return(hass);
}
```

3.4 Verbindung umlenken

Ankommende Verbindungen für eine lokale CMX-Anwendung erhält zunächst der Prozeß, der als erster diese Anwendung angemeldet hat. Um nun z.B. bestimmte Verbindungen bestimmten Prozessen zuordnen zu können, kann man eine Verbindung an einen anderen Prozeß weitergeben. Man kann natürlich auch aktiv aufgebaute Verbindungen weitergeben.

Beide Prozesse müssen zur selben Anwendung gehören, das heißt, sich mit demselben LOKALEN NAMEN angemeldet haben. Sie brauchen aber nicht verwandt zu sein. Der empfangende Prozeß muß die Bereitschaft, eine Verbindungsumlenkung entgegenzunehmen, beim Anmelden an CMX mitteilen.

Ablauf beim Umlenken einer Verbindung

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

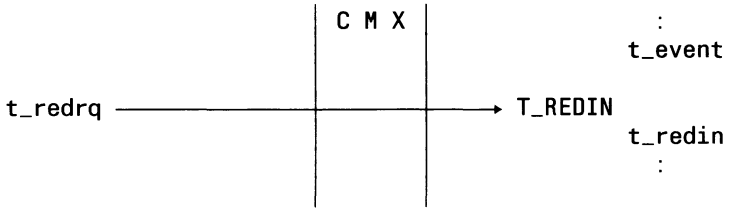


Bild 3-6 Verbindung umlenken

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

Soll die CMX-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen Sie anstatt t_redrq(), t_redin() und t_event die Aufrufe x_redrq(), x_redin() und x_event() verwenden. Der Prozeß B erhält das Ereignis X_REDIN zugestellt.

3.5 Beispiel zur Umlenkung einer Verbindung

Die folgenden Programme zeigen, wie man eine Verbindung umlenken kann. Das Programm redir_a baut aktiv eine Verbindung zu redir_p auf, wo diese Verbindung an einen Kindprozeß 1 umgelenkt wird.

Beispiele zur Verbindungsumlenkung mit ICMX(NEA) finden Sie bei der Beschreibung der entsprechenden Aufrufe im Abschnitt 6.3.

```
/*
 * CMX Test Command Source File
 *
 * Testprogramm fuer REDIRECT Primitive in CMX
 *
 * Modell:
 * Ein Vaterprozeß V fork'ed einen Sohn Prozeß S2.
 * V und S2 melden sich unter dem Namen 'REDIR__V', S1 unter dem
 * Namen 'REDIR__S' bei CMX an.
 * S1 baut eine Verbindung zu V auf, die dieser an S2 umlenkt. S2
 * beginnt mit dem Senden von Daten an S1. Im folgenden tauschen S1
 * und S2 ueber diese Verbindung eine Datenmenge (nicht laengere als
 * eine TIDU) aus, inzwischen wartet V auf die Beendigung seiner Sohnes.
 * Die Verarbeitung ist TS-Ereignis-gesteuert.
 *
 * Alle Ausgabe erfolgen auf stderr, Fehlermeldungen unbedingt, andere
 * Meldungen abhaengig von -d Option. Vater und Sohn melden stderr um in
 * Dateien
 *
 * Aufruf:
 *      redir [-d] [-lmsgsize] [-nmsgcnt] [-sson] [-pparent]
 *              d       debugging, mehrere 'd'  -> mehr Informationen
 *                      d       Verbindungsphase
 *                      dd      + Kommunikationsphase
 *              l       Nachrichtenlaenge msgsize <= 3072 (default: 3072)
 *                      nicht laengere als eine TIDU.
 *              n       Nachrichtenzahl msgcnt (default: 1000)
 *              s       Anwendungsname son von Sohn1 (default: "REDIR__S")
 *              p       Anwendungname parent von Vater und Sohn2
 *                      (default: "REDIR__V")
 */
static char sccsid[] = "@(#)redir_a.c   2.5 87/03/13 (DF154)";

#include        <stdio.h>
#include        <signal.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <errno.h>
#include        <sys/types.h>
#include        <sys/timeb.h>

#define NIL     0
#define TRUE    1
#define S1      "S1:"
#define FALSE   0
#define MAXLNG  3072
#define MAXCNT  1000
#define S1SEC   2
#define VSEC    2

char    s_buf[MAXLNG];          /* Nachrichtenpuffer fuer Datenaustausch */
int     s_bufl;                 /* Uebertragungslaenge */

int     tidul;                  /* TIDU-Laenge */
int     msgsize = MAXLNG;       /* gewuenschte Nachrichtenlaenge */
int     msgcnt = MAXCNT;        /* Zu uebertragene Nachrichten */
int     currcnt;                /* laufende Nachrichtennummer */
int     pid ;

int     datastop;
int     tref;
int     chain;                  /* TSDU-Indikator fuer t_datarq(),
                                   t_datain() */
```

```
int     mode;                   /* mode fuer t_event() */
char    rest[9];                /* Entaelt die Rechnerzeit in
                                   in <min>:<sek>.<millisek>,  */
                                /* Stunden werden wie Ruekkehr der
                                   rechzeit() geliefert  */

/*
 * Struktur der t_info()-Option
 */
struct  t_opti1  option = {T_OPTI1 };

/*
 * Struktur fuer Adressierung im TRANSDATA-Format
 * Verwendet um eine Verbindung aufzubauen
 */
#define SON     "REDIR__S"
#define PARENT  "REDIR__V"

char myname[TS_LPN+1] = { SON } ;
char pname[TS_LPN+1] = { PARENT } ;
union   t_address myaddress;
union   t_address partaddress;

/*
 * Optionen bei t_attach() fuer Vater und Soehne
 */
struct  t_opta1  son1opt = { T_OPTA1, T_ACTIVE, 1 };
struct  t_opta1  son2opt = { T_OPTA1, T_REDIRECT, 1 };
struct  t_opta1  paropt = { T_OPTA1, T_PASSIVE, 1 };

int     debug = 0;
extern  int     errno;
int     interrupt();            /* Routine zum Abfangen von SIGINT */

        /* PARAMETERSTRUKTUREN FUER DIRECTORY SYSTEM */
        /* ========================================= */

                /* Standardkopf */
Ts_head       ts_head = { TS_1VER01 } ;

        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Properties" */
        /* ----------------------------------------------------- */

Ts_leaf_name ts_myname   = { 1, TS_PN, NULL, myname } ;
Ts_leaf_name ts_partname = { 1, TS_PN, NULL, pname } ;

Ts_property ts_prop[] = { NULL, TS_ITEM, NULL, NULL,
                          TS_EMPTYPROP };

Ts_P13 ts1003 = { &ts_head,             /* Standardkopf                    */
                  TS_CMX_DIR_ID,        /* Directory-Identifier            */
                  NULL,                 /* Objektname                      */
                  NULL,                 /* irrelevanter Parameter          */
                  NULL,                 /* Werteliste fuer Eigenschaften   */
                                        /* Laenge der Werteliste           */
                  sizeof (union t_address),
                  ts_prop,              /* Liste der gewuenschten Eigensch. */
                  NULL, NULL, NULL } ; /* irrelevanter Parameter          */

/* ------------------------------------------------------------------------ */

main(argc, argv)
int argc;
```

```
char *argv[];
{
        register        i;
        char    errfil[20];
        int     pid;
        int     reason;

        /*
         * Optionsanalyse
         */
        while (--argc && **++argv == '-') {
                switch (*++*argv) {
                case 'd':
                        do {
                                debug++;
                        } while (*++*argv) ;
                        break;
                case 'l':
                        msgsize = atoi(++*argv);
                        break;
                case 'n':
                        msgcnt = atoi(++*argv);
                        break;
                case 's':
                        strcpy(myname, ++*argv);
                        break;
                case 'p':
                        strcpy(pname, ++*argv);
                        break;
                }
        }
        /*
         * SIGINT abfangen um den "stream" stderr auszugeben
         */
        signal(SIGINT, interrupt);
        /*
         * Datenpuffer initialisieren
         */
        for (i = 0; i < msgsize; i++)
                s_buf[i] = '*';
        datastop = FALSE;
        /*
         * stderr ummelden, Anmelden bei CMX
         */
        pid = getpid();
        sprintf(errfil, "SON1.%05d", pid);
        freopen(errfil, "w", stderr);
        setbuf(stderr,NULL);
        fprintf(stderr, "msgsize %d, msgcnt %d, myname '%s', pname '%s'\n",
                        msgsize, msgcnt, myname, pname);
        fprintf(stderr, "S1: %02d:%s  pid: %d\n", rechzeit(rest), rest, pid);

        ts1003.ts13leaf_name = &ts_myname ;
        ts_myname.ts_lf[0].ts_np_length = strlen (myname) ;
        ts1003.ts13prval = &myaddress ;
        ts_prop[0].ts_name = TS_LNAME ;
        if (ts13_read_properties (&ts1003) == TS_ERROR) {
                fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                        ts_head.ts_retcode, ts_head.ts_errclass,
                        ts_head.ts_errvalue) ;
                exit (1) ;
        } else {
                if (ts_prop[0].ts_length == 0) {
```

```
                fprintf (stderr, "TS-Fehler (Eigenschaft
                'eigener Name' nicht gefunden)\n") ;
                exit (1) ;
        }
}

if (t_attach (&myaddress, &son1opt) == T_ERROR) {
        fprintf(stderr, "S1: Fehler %d bei t_attach '%s'\n",
        t_error(), myname);
        exit(1);
}
fprintf(stderr, "S1: %02d:%s  angemeldet als '%s'\n",
rechzeit(rest), rest, myname);

/*
 * Verbindung aufbauen
 */
ts1003.ts13leaf_name = &ts_partname ;
ts_partname.ts_lf[0].ts_np_length = strlen (pname) ;
ts1003.ts13prval = &partaddress ;
ts_prop[0].ts_name = TS_TRANS ;
if (ts13_read_properties (&ts1003) == TS_ERROR) {
        fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                ts_head.ts_retcode, ts_head.ts_errclass,
                ts_head.ts_errvalue) ;
        exit (1) ;
} else {
        if (ts_prop[0].ts_length == 0) {
                fprintf (stderr, "TS-Fehler (Eigenschaft
                'Partneradresse' nicht gefunden)\n") ;
                exit (1) ;
        }
}
if (t_conrq(&tref, &partaddress, &myaddress, NIL) == T_ERROR) {
        fprintf(stderr, "S1: Fehler %d bei t_conrq zu '%s'\n",
                t_error(), pname);
        abbau(S1, FALSE);
}
if (debug)
        fprintf(stderr, "S1: %02d:%s  Verbindungsaufbau zu
        '%s' angefordert, tref %x\n",
                rechzeit(rest), rest, pname, tref);

/*
 * Ereignisgesteuerte Verarbeitung:
 * i.a. t_event() synchron (T_WAIT) wartend, fuer Datenausgabe
 * jedoch asynchron (T_CHECK)
 */
mode = T_WAIT;
for (;;) {
        switch (t_event(&tref, mode, NIL)) {

        case T_CONCF:
                /*
                 * CONNECT CONFIRMATION
                 */
                if (t_concf(&tref, NIL) == T_ERROR) {
                        fprintf(stderr, "S1: Fehler %d bei
                        t_concf, tref %x\n",
                                t_error(), tref);
                        abbau(S1, TRUE);
                }
                if (debug) {
```

```
                fprintf(stderr, "S1: %02d:%s  Ver-
                bindung zu '%s', tref %x aufgebaut\n",
                        rechzeit(rest), rest, pname, tref);
        }
        if (t_info(&tref, &option) == T_ERROR) {
                fprintf(stderr, "S1: Fehler %d bei t_info\n",
                        t_error());
                abbau(S1, TRUE);
        }
        if((tidul = option.t_maxl) < msgsize) {
                fprintf(stderr,"S1: %02d:%s  Diese
                Version gestattet nur Nachrichten \n",
                rechzeit(rest),rest);
                fprintf(stderr,"             nicht
                nicht laengere als eine TIDU (%d Bytes).
                \n", tidul);
                abbau(S1, TRUE);
        }
        break;

case T_ERROR:
        fprintf(stderr, "S1: Fehler %d bei t_event\n",
                t_error());
        abbau(S1, TRUE);
        break;

case T_DATAIN:
        /*
         * Datenankunft vom Sohn2. Nachricht annehmen
         * und beim naechsten T_NOEVENT "reflektieren".
         */
        s_bufl = tidul;
        if (t_datain(&tref, s_buf, &s_bufl, &chain)
           == T_ERROR){
                fprintf(stderr, " S1: Fehler %d bei
                t_datain, tref %x\n",
                        t_error(), tref);
                abbau(S1, TRUE);
        }
        if (chain == T_END) {
                /*
                 * Nachricht abgeholt
                 */
                currcnt++;
                if (debug > 1)
                        fprintf(stderr, "S1: %02d:%s
                        Nachricht %d empfangen\n",
                                rechzeit(rest), rest,
                                currcnt);
                mode = T_CHECK; /* erzwingt T_NOEVENT */
        }
        break;

case T_NOEVENT:
        if (mode == T_WAIT) {
                /*
                 * blockierender t_event unterbrochen
                   fortsetzen
                 */
                if (debug)
                        fprintf(stderr, "S1: t_event
                        (,T_WAIT,) unterbochen\n");
                break;
```

```
                        }
                        /*
                         * Nachricht an Sohn2 zuruecksenden, dann auf
                         * T_DATAIN warten.
                         */
                        if (send(S1)) {
                                if (debug > 1)
                                        fprintf(stderr, "S1: %02d:%s
                                        Nachricht %d gesendet\n",
                                                rechzeit(rest), rest,
                                                currcnt);
                                mode = T_WAIT;  /* Sohn2 wird nun senden */
                        }
                        break;

                case T_DISIN:
                        /*
                         * DISCONNECT INDICATION: Verbindungsabbau
                         * durch System oder Vater, weil dieser den
                         * CONNECT REQUEST nicht akzeptieren kann
                         */
                        if (t_disin(&tref, &reason, NIL) == T_ERROR) {
                                fprintf(stderr, "S1: Fehler %d bei
                                t_disin, tref %x\n",
                                        t_error(), tref);
                        }
                        fprintf(stderr, "S1: %02d:%s  Nachricht %d von
                        %d empfangen\n",
                                rechzeit(rest), rest, currcnt, msgcnt);
                        if (debug)
                                fprintf(stderr, "S1: %02d:%s  Verbin-
                                dungsabbau tref %x empfangen, reason %d\n",
                                        rechzeit(rest), rest, tref, reason);
                        if (t_detach(&myaddress) == T_ERROR) {
                                fprintf(stderr, "S1: Fehler %d bei
                                t_detach '%s'\n",
                                        t_error(), myname);
                        }
                        fprintf(stderr, "S1: %02d:%s  Anwendung '%s'
                        abgemeldet\n",
                                rechzeit(rest), rest, myname);
                        exit(0);
                        break;

                case T_DATAGO:
                        datastop = FALSE;
                        break;

                default:
                        fprintf(stderr, "S1: Unerwarteter event\n");
                        break;
                }
        }
}

/*
 * Verbindungsabbau und Abmeldung bei CMX nach Fehlern
 * who gibt den Aufrufer an (Vater, Sohn1, Sohn2)
 * how gibt an, ob die Verbindung abgebaut werden
 * soll (TRUE) oder nicht (FALSE)
 */
abbau(who, how)
char *who;
```

```
{
        if (how) {
                if (t_disrq(&tref, NIL) == T_ERROR)
                        fprintf(stderr, "%s:  Fehler %d bei t_disrq,
                        tref %x\n",
                                who, t_error(), tref);
                if (debug)
                        fprintf(stderr, "%s:  %02d:%s  Verbindung
                        tref %x abgebaut\n",
                                who, rechzeit(rest), rest, tref);
        }
        if (t_detach(&myaddress) == T_ERROR)
                fprintf(stderr, "%s: Fehler %d bei t_detach '%s'\n",
                         who, t_error(), myname);
        fprintf(stderr, "%02d:%s   Anwendung '%s' abgemeldet\n",
                rechzeit(rest), rest, myname);
        exit(how);
}

/*
 * Nachricht an Partner senden
 * return(TRUE), falls gesendet, return(FALSE), falls nicht, weil T_DATASTOP
 */
send(who)
char *who;
{
        int             rc;

        if (datastop) {
                mode = T_CHECK;
                return(FALSE);
        }
        chain = T_END;
        s_bufl = msgsize;
        if ((rc = t_datarq(&tref, s_buf, &s_bufl, &chain)) == T_ERROR) {
                fprintf(stderr, "%s Fehler %d bei t_datarq, tref %x\n",
                        who, t_error(), tref);
                        abbau();
        }
        if (rc == T_DATASTOP)
                datastop = TRUE;
        return(TRUE);
}

/*
 * interrupt
 * Abfangen von DEL: ausgeben streams
 */
interrupt()
{
        fflush(stderr);
        fclose(stderr);
        exit(1);
}
/* ========================================================================
 * Die Funktion rechnet die absolute Zeit, in der die
 * bestimmte Tat ausgefuert ist.
 *  EINGABE: 'wert' Pointer zu einem Feld wo der Zeit ist.
 *  AUSGABE: 'hass' Stundewert
 */
rechzeit(wert)
char *wert;
{
```

```
        struct timeb pt;
        int hass, mass, dec;

        ftime(&pt);
        if((hass = pt.time/3600%24 - pt.timezone/60) == 24)     /* Stunden */
                hass = 0;
        mass = pt.time%3600/60;          /* Minuten */
        dec = pt.time%3600%60;           /* Sekunden */
        sprintf(wert,"%02d:%02d.%03u", mass, dec, pt.millitm);
        return(hass);
}
```

```
/*
 * CMX Test Command Source File
 *
 * Testprogramm fuer REDIRECT Primitive in CMX
 *
 * Modell:
 * Ein Vaterprozeß V fork'ed einen Sohn Prozeß S2.
 * V und S2 melden sich unter dem Namen 'REDIR__V' bei CMX an.
 * V wartet auf einen Verbindungsaufbauwunsch von 'REDIR__S' (im
 * folgenden S1 genannt) und lenkt die Verbindung an S2 um.
 * S2 beginnt mit dem Senden von Daten an S1. Im folgenden tauschen S1
 * und S2 ueber diese Verbindung eine Datenmenge (nicht laengere
 * als eine TIDU) aus, inzwischen wartet V auf die Beendigung
 * seines Sohnes. Die Verarbeitung ist TS-Ereignis-gesteuert.
 *
 * Alle Ausgabe erfolgen auf stderr, Fehlermeldungen unbedingt, andere
 * Meldungen abhaengig von -d Option. Vater und Sohn melden stderr um in
 * Dateien
 *
 * Aufruf:
 *      redir [-d] [-lmsgsize] [-nmsgcnt] [-sson] [-pparent]
 *              d       debugging, mehrere 'd' -> mehr Informationen
 *                      d       Verbindungsphase
 *                      dd      + Kommunikationsphase
 *              l       Nachrichtenlaenge msgsize <= 3072 (default: 3072)
 :                      nicht laengere als eine TIDU.
 *              n       Nachrichtenzahl msgcnt (default: 1000)
 *              p       Anwendungsname son von Sohn1 (default: "REDIR__S")
 *              s       Anwendungname parent von Vater und Sohn2
 *                      (default: "REDIR__V")
 */
static char sccsid[] = "@(#)redir_p.c   2.5 87/03/13 (DF154)";

#include        <stdio.h>
#include        <signal.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <errno.h>
#include        <sys/types.h>
#include        <sys/timeb.h>

#define NIL     0
#define TRUE    1
#define S2      "S2:"
#define VATER   "V:"
#define FALSE   0
#define MAXLNG  3072
#define MAXCNT  1000
#define S1SEC   2
#define VSEC    2

char    s_buf[MAXLNG];          /* Nachrichtenpuffer fuer Datenaustausch */
int     s_bufl;                 /* Uebertragungslaenge */

int     tidul;                  /* TIDU-Laenge */
int     msgsize = MAXLNG;       /* gewuenschte Nachrichtenlaenge */
int     msgcnt = MAXCNT;        /* Zu uebertragene Nachrichten */
int     currcnt;                /* laufende Nachrichtennummer */

int     datastop;
```

```
int     tref;
int     chain;                  /* TSDU-Indikator fuer t_datarq(),
                                   t_datain() */
int     mode;                   /* mode fuer t_event() */
char    rest[9];                /* Enthaelt die Rechnerzeit in
                                   <min>:<sek>.<millisek>,   */
                                /* die Stunden werden wie Ruekkehr
                                   der rechzeit() geliefert.  */

int     pidson2;                /* pid von Sohn2 nach fork() */
int     pidpar;                 /* pid des Vaters nach t_redin() */

int     status;                 /* fuer wait() */

/*
 * Struktur der t_info()-Option
 */
struct  t_opti1  option = {T_OPTI1 };

/*
 * Struktur fuer Adressierung im TRANSDATA-Format
 * Verwendet um eine Verbindung aufzubauen
 */
#define SON     "REDIR__S"
#define PARENT  "REDIR__V"

char myname[TS_LPN+1] = { PARENT } ;
char pname[TS_LPN+1] = { SON } ;
union   t_address myaddress;
union   t_address partaddress;

/*
 * Optionen bei t_attach() fuer Vater und Sohn
 */
struct  t_opta1  son2opt = { T_OPTA1, T_REDIRECT, 1 };
struct  t_opta1  paropt = { T_OPTA1, T_PASSIVE, 1 };

int     debug = 0;
extern  int     errno;
int     interrupt();            /* Routine zum Abfangen von SIGINT */

        /* PARAMETERSTRUKTUREN FUER DIRECTORY SYSTEM */
        /* ========================================= */

                /* Standardkopf */
Ts_head       ts_head = { TS_1VER01 } ;

        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Leafs" */
        /* ------------------------------------------------ */

Ts_leaf_name ts_sname = { 0 } ;
Ts_leaf_name ts_pname ;
char         ts_nlist[TS_LPN] ;
Ts_filter ts_filter = { TS_TRANS, NULL, (char *) &partaddress } ;

Ts_P11 ts1001 = { &ts_head,         /* Standardkopf                 */
                  TS_CMX_DIR_ID,    /* Directory-Identifier         */
                  &ts_sname,        /* Suchkriterium (Name)         */
                  TS_GENERAL,       /* Suchmodus                    */
                  &ts_filter,       /* Suchkriterium (Eigenschaft)  */
```

```
                TS_LOCAL,          /* Suchmodus                      */
                NULL, NULL,        /* irrelevante Parameter          */
                ts_nlist,          /* Werteliste fuer Namensteile    */
                sizeof(ts_nlist), /* Laenge der Werteliste          */
                1,                 /* Anzahl der erwarteten Namen    */
                &ts_pname,         /* Liste fuer gefundene Objekte   */
                NULL } ;           /* Anzahl der gefundenen Objekte  */
                                   /* (Rueckgabeparameter)           */

        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Properties" */
        /* ------------------------------------------------------ */

Ts_leaf_name ts_myname   = { 1, TS_PN, NULL, myname } ;
Ts_leaf_name ts_partname = { 1, TS_PN, NULL, pname } ;

Ts_property ts_prop[] = { NULL, TS_ITEM, NULL, NULL,
                          TS_EMPTYPROP };

Ts_P13 ts1003 = { &ts_head,            /* Standardkopf                   */
                  TS_CMX_DIR_ID,       /* Directory-Identifier           */
                  NULL,                /* Objektname                     */
                  NULL,                /* irrelevanter Parameter         */
                  NULL,                /* Werteliste fuer Eigenschaften  */
                                       /* Laenge der Werteliste          */
                  sizeof (union t_address),
                  ts_prop,             /* Liste der gewuenschten Eigensch. */
                  NULL, NULL, NULL } ; /* irrelevanter Parameter         */

/* ------------------------------------------------------------------------ */

main(argc, argv)
int argc;
char *argv[];
{
        register        i;
        char    errfil[20];
        int     pid;
        int     reason;

        /*
         * Optionsanalyse
         */
        while (--argc && **++argv == '-') {
                switch (*++*argv) {
                case 'd':
                        do {
                                debug++;
                        } while (*++*argv) ;
                        break;
                case 'l':
                        msgsize = atoi(++*argv);
                        break;
                case 'n':
                        msgcnt = atoi(++*argv);
                        break;
                case 's':
                        strcpy(myname, ++*argv);
                        break;
                case 'p':
                        strcpy(pname, ++*argv);
```

```
                break;
        }
}
/*
 * SIGINT abfangen um den "stream" stderr auszugeben
 */
signal(SIGINT, interrupt);
/*
 * Datenpuffer initialisieren
 */
for (i = 0; i < msgsize; i++)
        s_buf[i] = '*';
datastop = FALSE;
/*
 * Sohn forken
 */
if ((pidson2 = fork()) == 0) {
        /*
         * Hier in Sohn2: Anmelden bei CMX
         */
        pid = getpid();
        sprintf(errfil, "SON2.%05d", pid);
        freopen(errfil, "w", stderr);
        setbuf(stderr,NULL);
        fprintf(stderr, "msgsize %d, msgcnt %d, myname '%s',
        pname '%s'\n",
                msgsize, msgcnt, myname, pname);
        fprintf(stderr, "S2: %02d:%s  pid: %d\n", rechzeit(rest),
        rest, pid);
        ts1003.ts13leaf_name = &ts_myname ;
        ts_myname.ts_lf[0].ts_np_length = strlen (myname) ;
        ts1003.ts13prval = &myaddress ;
        ts_prop[0].ts_name = TS_LNAME ;
        if (ts13_read_properties (&ts1003) == TS_ERROR) {
                fprintf (stderr, "TS-Fehler (Typ: %d Klasse:
                %d Wert: %d)\n",
                        ts_head.ts_retcode, ts_head.ts_errclass,
                        ts_head.ts_errvalue) ;
                exit (1) ;
        } else {
                if (ts_prop[0].ts_length == 0) {
                        fprintf (stderr, "TS-Fehler (Eigenschaft
                        'eigener Name' nicht gefunden)\n") ;
                        exit (1) ;
                }
        }

        if (t_attach(&myaddress, &son2opt) == T_ERROR) {
                fprintf(stderr, "S2: Fehler %d bei t_attach '%s'\n",
                        t_error(), myname);
                exit(1);
        }
        fprintf(stderr, "S2: %02d:%s  angemeldet als '%s'\n",
                rechzeit(rest), rest, myname);

        /*
         * Ereignisgesteuerte Verarbeitung:
         * i.a. t_event() synchron (T_WAIT) wartend, fuer Daten-
         * ausgabe, jedoch asynchron (T_CHECK)
         */
```

```
mode = T_WAIT;
for (;;) {
        switch (t_event(&tref, mode, NIL)) {

        case T_REDIN:
                /*
                 * Verbindungsumlenkung durch Vater
                 * Verbindung annehmen, mit Senden an Sohn1
                 * beginnen.
                 */
                if (t_redin(&tref, &pidpar, NIL) ==
                T_ERROR) {
                        fprintf(stderr, "S2: Fehler %d
                        bei t_redin, tref %x, pid %x\n",
                                t_error(), tref, pidpar);
                        abbau(S2, TRUE);
                }
                if (debug) {
                        fprintf(stderr, "S2: %02d:%s
                        Verbindung tref %x erhalten von %d\n",
                                rechzeit(rest), rest, tref,
                                pidpar);
                }
                if (t_info(&tref, &option) == T_ERROR) {
                        fprintf(stderr, "S2: Fehler %d
                        bei t_info\n",
                                t_error());
                        abbau(S2, TRUE);
                }
                if((tidul = option.t_maxl) < msgsize)  {
                        fprintf(stderr,"S2: %02d:%s
                        Diese Version gestattet nur Nach-
                        richten\n", rechzeit(rest), rest);
                        fprintf(stderr,"             nicht
                        laengere als eine TIDU (%d Bytes).
                        \n", tidul);
                        abbau(S2, TRUE);
                }
                mode = T_CHECK; /* fuer Senden bei
                T_NOEVENT */
                break;

        case T_NOEVENT:
                if (mode == T_WAIT) {
                        /*
                         * blockierender t_event unterbrochen
                         * fortsetzen
                         */
                        if (debug)
                                fprintf(stderr, "S2:
                                t_event(,T_WAIT,) unter-
                                brochen\n");
                        break;
                }

                /*
                 * Naechste Nachricht senden an Sohn1,
                 * dann auf T_DATAIN warten.
                 */
                if (currcnt == msgcnt) {
```

```
                fprintf(stderr, "S2: %02d:%s  Nach-
                richt %d von %d gesendet\n",
                        rechzeit(rest), rest,
                        currcnt, msgcnt);
                abbau(S2, TRUE);
        }
        if (send(S2)) {
                currcnt++;
                if (debug > 1)
                        fprintf(stderr, "S2: %02d:%s
                        Nachricht %d gesendet\n",
                        rechzeit(rest), rest, currcnt);
                mode = T_WAIT;
        }
        break;

case T_DATAIN:
        /*
         * Datenankunft vom Sohn1, "reflektierte"
         * Nachricht. Beim naechsten T_NOEVENT
         * weitersenden.
         */
        s_bufl = tidul;
        if (t_datain(&tref, s_buf, &s_bufl,
        &chain) == T_ERROR){
                fprintf(stderr, "S2: Fehler %d bei
                t_datain, tref %x\n",
                        t_error(), tref);
                abbau(S2, TRUE);
        }
        if (chain == T_END) {
                /*
                 * Nachricht abgeholt
                 */
                if (debug > 1)
                        fprintf(stderr, "S2: %02d:%s
                        Nachricht %d empfangen\n",
                                rechzeit(rest), rest,
                                currcnt);
                mode = T_CHECK; /* erzwingt
                T_NOEVENT */
        }
        break;

case T_DISIN:
        if (t_disin(&tref, &reason, NIL) == T_ERROR) {
                fprintf(stderr, "S2: Fehler %d bei
                t_disin tref %x\n",
                        t_error(), tref);
        }
        if (debug)
                fprintf(stderr, "S2: %02d:%s  Ver-
                bindungsabbau tref %x empfangen,
                reason %d\n",
                        rechzeit(rest), rest, tref,
                        reason);
        if (t_detach(&myaddress) == T_ERROR) {
                fprintf(stderr, "S2: Fehler %d bei
                t_detach '%s'\n",
                        t_error(), myname);
```

```
                        }
                        fprintf(stderr, "S2: %02d:%s  Anwendung '%s'
                        abgemeldet\n",
                                rechzeit(rest), rest, myname);
                        exit(0);
                        break;

                case T_DATAGO:
                        datastop = FALSE;
                        break;

                case T_ERROR:
                        fprintf(stderr, "S2: Fehler %d bei t_event\n",
                                t_error());
                        abbau(S2, TRUE);
                        break;

                default:
                        fprintf(stderr, "S2: Unerwarteter event\n");
                        break;
                }
        }
}
/*
 * Hier wieder im Vater:
 * Anmelden bei CMX, Verbindungsaufbau durch S1 abwarten
 */
pid = getpid();
sprintf(errfil, "PARENT.%05d", pid);
freopen(errfil, "w", stderr);
setbuf(stderr,NULL);
fprintf(stderr, "msgsize %d, msgcnt %d, myname '%s', pname '%s'\n",
                msgsize, msgcnt, myname, pname);
fprintf(stderr, "V: %02d:%s  pid: %d\n", rechzeit(rest), rest, pid);

ts1003.ts13leaf_name = &ts_myname ;
ts_myname.ts_lf[0].ts_np_length = strlen (myname) ;
ts1003.ts13prval = &myaddress ;
ts_prop[0].ts_name = TS_LNAME ;
if (ts13_read_properties (&ts1003) == TS_ERROR) {
        fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                ts_head.ts_retcode, ts_head.ts_errclass,
                ts_head.ts_errvalue) ;
        exit (1) ;
} else {
        if (ts_prop[0].ts_length == 0) {
                fprintf (stderr, "TS-Fehler (Eigenschaft
                'eigener Name' nicht gefunden)\n") ;
                exit (1) ;
        }
}
if (t_attach(&myaddress, &paropt) == T_ERROR) {
        fprintf(stderr, "V: Fehler %d bei t_attach '%s'\n",
                t_error(), myname);
        exit(1);
}
fprintf(stderr, "V: %02d:%s  angemeldet als '%s'\n", rechzeit(rest),
rest, myname);

mode = T_WAIT;
```

```
switch (t_event(&tref, mode, NIL)) {

case T_CONIN:
        /*
         * Verbindungsaufbauwunsch akzeptieren
         */
        if (t_conin(&tref, &myaddress, &partaddress, NIL) == T_ERROR){
                fprintf(stderr, "V: Fehler %d bei t_conin\n",
                        t_error());
                abbau(VATER, TRUE);
        }
        if (t_conrs(&tref, NIL) == T_ERROR) {
                fprintf("V: Fehler %d bei t_conrs, tref %x\n",
                        t_error(), tref);
                abbau(VATER, TRUE);
        }
        if (debug) {
                ts_filter.ts_pr_length = partaddress.tpartaddr.
                t_palng ;
                if (ts11_read_leafs (&ts1001) == TS_ERROR) {
                        fprintf (stderr, "TS-Fehler (Typ: %d Klasse:
                        %d Wert: %d)\n",
                                ts_head.ts_retcode, ts_head.
                                ts_errclass, ts_head.ts_errvalue) ;
                        exit (1) ;
                }
                if (ts1001.ts11real_no != 1)
                        fprintf (stderr, "TS-Fehler (Partnername
                        nicht zu ermitteln)") ;
                ts_nlist[ts_pname.ts_lf[0].ts_np_length] = '\0' ;
                fprintf(stderr, "V: %02d:%s  Verbindung von '%s'
                akzeptiert, tref %x\n",
                        rechzeit(rest), rest, ts_nlist, tref);
        }
        /*
         * Verbindung an Sohn2 umlenken
         * nach VSEC Sekunden des Wartens
         */
        sleep(VSEC);
        if (t_redrq(&tref, &pidson2, NIL) == T_ERROR) {
                fprintf(stderr, "V: Fehler %d bei t_redrq, tref %x,
                pid %d\n",
                        t_error(), tref, pidson2);
                abbau(VATER, TRUE);
        }
        if (debug)
                fprintf(stderr, "V: %02d:%s  Verbindung tref %x
                umgelenkt an %d\n",
                                rechzeit(rest), rest, tref, pidson2);

        break;

default:
        fprintf(stderr, "V: Unerwarteter event\n");
        exit(1);
}
/*
 * Warten auf Beendigung der Soehne
 */
while ((pid = wait(&status)) != -1) {
```

```
                fprintf(stderr, "V: %02d:%s  Sohn pid %d beendet mit
                Status %x\n",
                        rechzeit(rest), rest, pid, status);
        }
        if (errno != ECHILD)
                fprintf(stderr, "Fehler %d bei wait()\n", errno);
        if (t_detach(&myaddress) == T_ERROR)
                fprintf(stderr, "%s Fehler %d bei t_detach '%s'\n",
                         VATER, t_error(), myname);
        fprintf(stderr, "V: %02d:%s  Anwendung '%s' abgemeldet\n",
                rechzeit(rest), rest, myname);

}

/*
 * Verbindungsabbau und Abmeldung bei CMX nach Fehlern
 * who gibt den Aufrufer an (Vater, Sohn1, Sohn2)
 * how gibt an, ob die Verbindung abgebaut werden soll (TRUE)
 * oder nicht (FALSE)
 */
abbau(who, how)
char *who;
{
        if (how) {
                if (t_disrq(&tref, NIL) == T_ERROR)
                        fprintf(stderr, "%s Fehler %d bei t_disrq, tref %x\n",
                                who, t_error(), tref);
                if (debug)
                        fprintf(stderr, "%s %02d:%s  Verbindung tref
                        %x abgebaut\n",
                                who, rechzeit(rest), rest, tref);
        }
        if (t_detach(&myaddress) == T_ERROR)
                fprintf(stderr, "%s Fehler %d bei t_detach '%s'\n",
                         who, t_error(), myname);
        fprintf(stderr, "%s %02d:%s  Anwendung '%s' abgemeldet\n",
                who, rechzeit(rest), rest, myname);
        exit(how);
}

/*
 * Nachricht an Partner senden
 * return(TRUE), falls gesendet, return(FALSE), falls nicht, weil T_DATASTOP
 */
send(who)
char *who;
{
        int             rc;

        if (datastop) {
                mode = T_CHECK;
                return(FALSE);
        }
        chain = T_END;
        s_bufl = msgsize;
        if ((rc = t_datarq(&tref, s_buf, &s_bufl, &chain)) == T_ERROR) {
                fprintf(stderr, "%s Fehler %d bei t_datarq, tref %x\n",
                        who, t_error(), tref);
                        abbau();
        }
```

```
        if (rc == T_DATASTOP)
                datastop = TRUE;
        return(TRUE);
}

/*
 * interrupt
 * Abfangen von DEL: ausgeben streams
 */
interrupt()
{
        fflush(stderr);
        fclose(stderr);
        exit(1);
}
/* ========================================================================== */
 * Die Funktion rechnet die absolute Zeit, in der die
 * bestimmte Tat ausgefuehrt ist.
 *  EINGABE: 'wert' Pointer zu einem Feld wo der Zeit ist.
 *  AUSGABE: 'hass' Stundewert
 */
rechzeit(wert)
char *wert;
{
        struct timeb pt;
        int hass, mass, dec;

        ftime(&pt);
        if((hass = pt.time/3600%24 - pt.timezone/60) == 24)     /* Stunden */
                hass = 0;
        mass = pt.time%3600/60;          /* Minuten */
        dec = pt.time%3600%60;           /* Sekunden */
        sprintf(wert,"%02d:%02d.%03u", mass, dec, pt.millitm);
        return(hass);
}
```

4 Daten übertragen

Partner A möchte an Partner B Daten senden. Die Gesamtmenge von Daten, die eine logische Einheit bilden, soll als Nachricht bezeichnet werden (der terminus technicus ist: TSDU = Transport Service Data Unit). Die Länge einer Nachricht ist beliebig (siehe jedoch Abschnitt 8.3).

CMX kann jedoch immer nur eine begrenzte Datenmenge auf einmal annehmen. Diese bezeichnet man als Dateneinheit (der terminus technicus ist: TIDU = Transport Interface Data Unit). Wie lang eine Dateneinheit höchstens sein darf, hängt vom Transportsystem ab. Die Länge muß man mit dem Aufruf t_info() bzw. x_info() bei ICMX(NEA) für jede Verbindung abfragen.

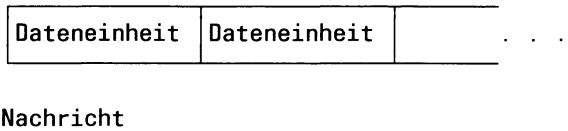


Bild 4-1 Dateneinheit und Nachricht

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

Wenn das Transportsystem diese Option bietet und beide Partner dies beim Verbindungsaufbau vereinbart haben, können sie auch Vorrangdaten austauschen. Vorrangdaten sind kleine Datenmengen, die mit "Vorrang" vor den gewöhnlichen Daten transportiert werden.

Bei Vorrangdaten gibt es keine Unterteilung in Dateneinheiten, sie können immer nur auf einmal übertragen werden.

4.1 Senden und Empfangen von gewöhnlichen Daten

Gewöhnliche Daten senden Sie mit dem Aufruf **t_datarq()**.
Jeder solche Aufruf sendet maximal eine Dateneinheit. Im einfachsten Fall ist der Ablauf so:

Der sendende Partner schickt mit jedem Aufruf eine Dateneinheit. Der empfangende Partner erhält das Ereignis T_DATAIN angezeigt. Das ist die Mitteilung, daß Daten angekommen sind. Der empfangende Partner muß die Daten mit dem Aufruf **t_datain()** annehmen.

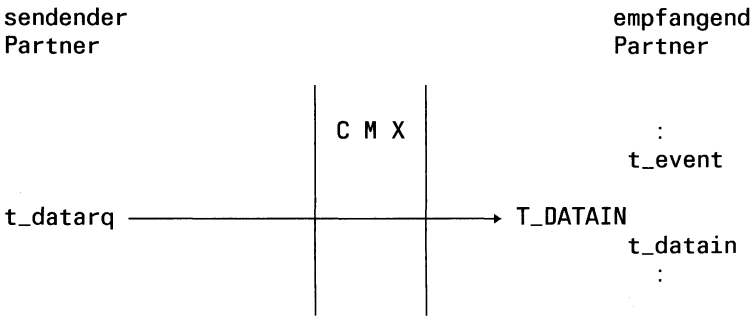


Bild 4-2 Daten übertragen

Wenn die Nachricht länger ist als eine Dateneinheit ...

muß sie in Dateneinheiten zerlegt werden. Das geht so:

Der sendende Partner bestimmt als Sender, wann die Nachricht zu Ende ist. Bei jedem Absenden einer Dateneinheit mit **t_datarq()** gibt er im Parameter chain an, ob weitere Dateneinheiten in dieser Nachricht folgen oder die gerade zu sendende die letzte ist.

Der empfangende Partner bekommt in gleicher Weise bei jedem Aufruf **t_datain()** mitgeteilt, ob noch weitere Dateneinheiten in dieser Nachricht folgen.

Jede Dateneinheit kündigt CMX mit einem Ereignis T_DATAIN an, aber:

Dateneinheit ist nicht gleich Dateneinheit!

Die Länge einer Dateneinheit kann bei beiden Partnern unterschiedlich sein. Deshalb kann es sein, daß der empfangende Partner weniger oft **t_datain()** aufrufen muß, als der sendende Partner **t_datarq()** (oder umgekehrt). Denn der empfangende Partner liest Dateneinheiten in "seiner" Länge. Das sieht dann wie folgt aus:

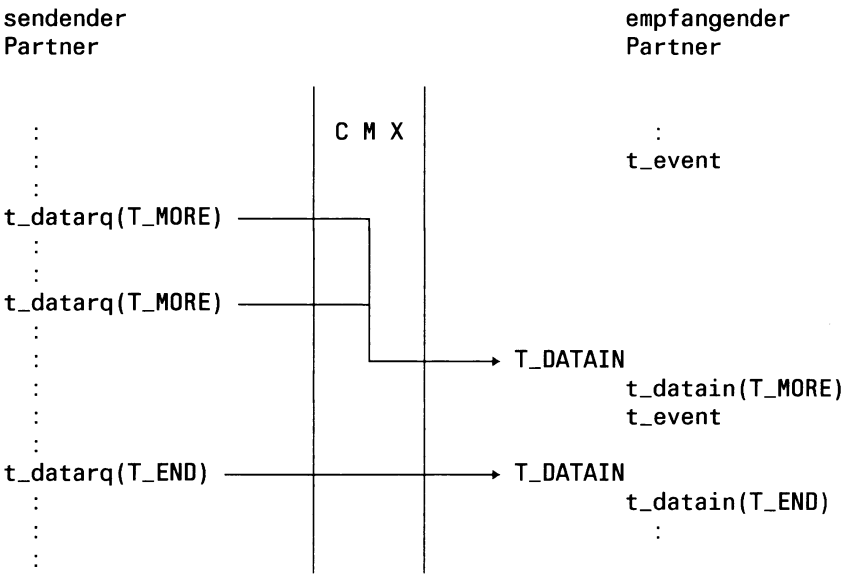


Bild 4-3 Nachricht in mehreren Dateneinheiten

Der Ergebniswert von t_datain()

Bei **t_datain()** müssen Sie eine Länge angeben, in der Sie die angekommenen Daten lesen wollen. Wenn die angegebene Länge kleiner ist, als die Dateneinheit beim Empfänger, können Sie die Daten eventuell nicht auf einmal lesen. Der Ergebniswert von **t_datain()** gibt Ihnen dann die Restlänge der Daten in der anstehenden Dateneinheit an.

Ist eine Dateneinheit noch nicht vollständig gelesen, müssen Sie erneut **t_datain()** aufrufen, solange, bis Sie die Dateneinheit vollständig gelesen haben. Währenddessen dürfen Sie auch nicht t_event() aufrufen, die Verbindung umlenken oder den Datenfluß regeln (siehe Abschnitt 2.5).

Beachten Sie, daß CMX nicht garantiert, daß beim empfangenden Partner alle Dateneinheiten (außer der letzten) einer Nachricht maximal gefüllt sind, selbst dann nicht, wenn die Dateneinheit beim sendenden und empfangenden Partner gleich ist und der sendende Partner nur maximal gefüllte Dateneinheiten sendet.

Soll die CMX-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen Sie anstatt der Aufrufe t_datarq(), t_datain() und t_event() die Aufrufe x_datarq(), x_datain() und x_event() benutzen. Als Ereignis wird x_DATAIN angezeigt, die Parameterwerte für die Nachrichtenlänge heißen X_MORE und X_END. Beachten Sie bitte, daß es bei ICMX(NEA) Einschränkungen bezüglich der Datenlängen gibt, insbesondere ist ein stückweises Lesen der Dateineinheit nicht möglich. Die entsprechenden Festlegungen finden Sie im Abschnitt 6.3.

4.2 Senden und Empfangen von Vorrangdaten

Wenn Sie beim Verbindungsaufbau (siehe Abschnitt 3.1) den Austausch von Vorrangdaten vereinbart haben, können Sie diesen wie folgt vornehmen.

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

> Der sendende Partner schickt mit einem Aufruf Vorrangdaten. Der empfangende Partner erhält das Ereignis T_XDATIN angezeigt. Das ist die Mitteilung, daß Vorrangdaten angekommen sind. Der empfangende Partner muß die Daten mit dem Aufruf **t_xdatin()** annehmen.

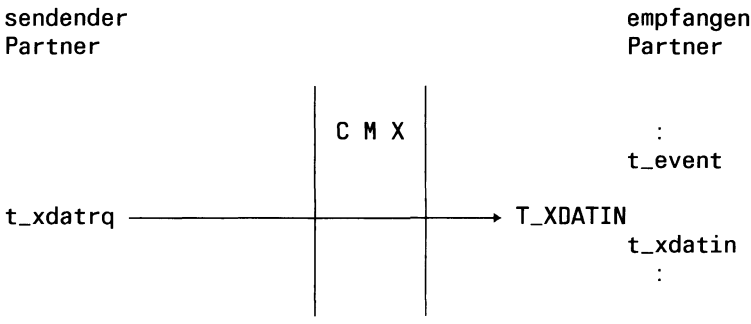


Bild 4-4 Vorrangdaten übertragen

Der Ergebniswert von t_xdatin()

Bei **t_xdatin()** müssen Sie eine Länge angeben, in der Sie die angekommenen Vorrangdaten lesen wollen. Wenn die angegebene Länge kleiner ist als die Anzahl der angekommenen Vorrangdaten, können Sie die Vorrangdaten nicht auf einmal lesen. Der Ergebniswert von **t_xdatin()** gibt Ihnen dann die Restlänge der anstehenden Vorrangdaten an.

Sind die Vorrangdaten noch nicht vollständig gelesen, müssen Sie erneut **t_xdatin()** aufrufen, solange bis Sie die Daten vollständig gelesen haben. Währenddessen dürfen Sie auch nicht t_event() aufrufen, die Verbindung umlenken oder den Datenfluß regeln (siehe Abschnitt 2.5).

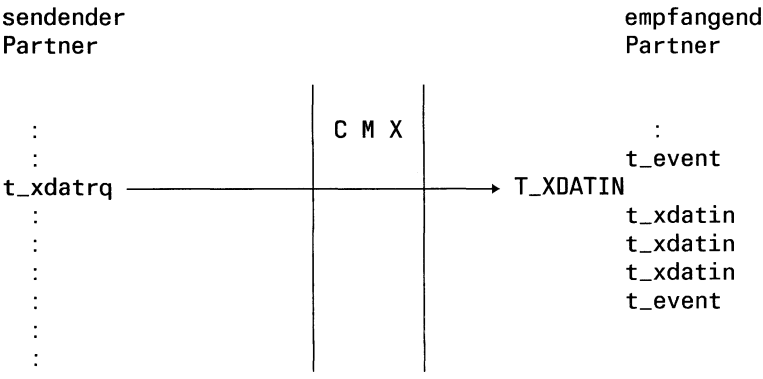


Bild 4-5 Vorrangdaten stückweise lesen

Soll die CMX-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen Sie anstatt der Aufrufe t_xdatrq(), t_xdatain() und t_event() die Aufrufe x_xdatrq(), x_xdatin() und x_event() benutzen. Als Ereignis wird X_XDATIN gemeldet. Bitte beachten Sie, daß es bei ICMX(NEA) Einschränkungen bezüglich der Datenlängen gibt. Insbesondere ist ein stückweises Lesen der Vorrangdaten nicht möglich. Die entsprechenden Festlegungen finden Sie im Abschnitt 6.3.

4.3 Flußregelung von Daten und Vorrangdaten

Wenn Sie nicht bereit sind, auf einer Verbindung Daten zu empfangen, dann teilen Sie dies CMX mit dem Aufruf **t_datastop()** mit. CMX stellt ab sofort das Ereignis T_DATAIN für diese Verbindung nicht mehr zu. Der Kommunikationspartner erhält von CMX bei **t_datarq()** das Ergebnis T_DATASTOP und darf keine weiteren Daten senden.

Sobald Sie wieder bereit sind, auf der Verbindung Daten zu empfangen, rufen Sie **t_datago()** auf. Der Kommunikationspartner erhält das Ereignis T_DATAGO zugestellt und Sie können wieder Daten von ihm empfangen. Sie erhalten das Ereignis T_DATAIN wieder zugestellt.

In gleicher Weise erfolgt die Flußregelung für Vorrangdaten. Sie verwenden dafür die Aufrufe **t_xdatstop()** und **t_xdatgo()**. Die entsprechenden Ereignisse dazu sind T_XDATIN und T_XDATGO.
Beachten Sie aber:

> Wenn Sie den Vorrangdatenfluß stoppen (mit **t_xdatstop()**), stoppt CMX implizit auch den Fluß für gewöhnliche Daten. Wenn Sie dann den Vorrangdatenfluß wieder freigeben (mit **t_xdatgo()**) bleibt der Fluß für gewöhnliche Daten gesperrt! Sie müssen ihn eigens freigeben (mit **t_datago()**).
>
> Bei Freigabe des Flusses gewöhnlicher Daten gibt CMX implizit auch den Fluß der Vorrangdaten wieder frei. Damit geben Sie nach einem Aufruf **t_xdatstop()** mit **t_datago()** sowohl den Fluß der gewöhnlichen als auch den Fluß der Vorrangdaten frei.

Was bringt es Ihnen, wenn Sie sich T_DATAIN oder T_XDATIN nicht mehr zustellen lassen?

Sie können inzwischen andere CMX-Aufrufe machen, z.B. eine weitere Verbindung aufbauen. Das wäre nicht möglich, wenn ein Ereignis T_DATAIN ansteht. Wenn Sie die Daten nicht abholen, meldet jeder Aufruf t_event() wieder das Ereignis T_DATAIN und Sie könnten nicht das zum Verbindungsaufbau erforderliche Ereignis T_CONCF erhalten.

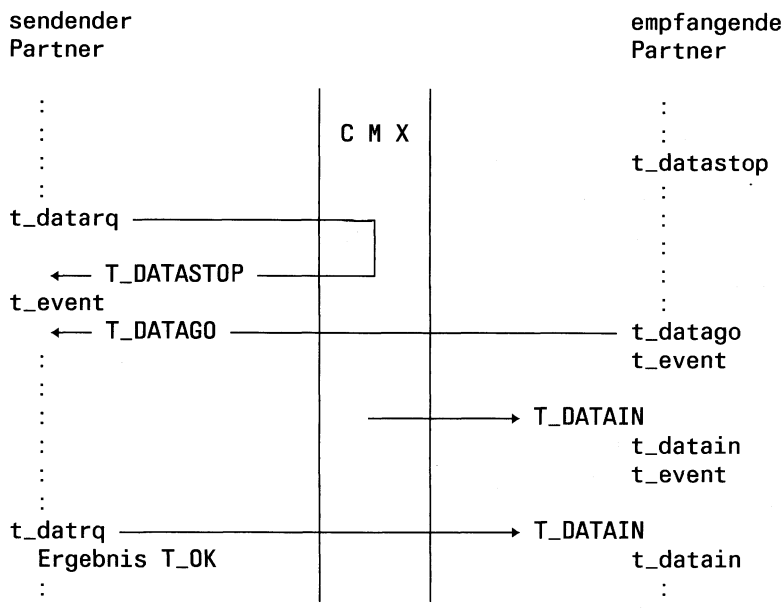


Bild 4-6 Datenflußregelung beim Sender

Der sendende Partner erhält T_DATASTOP als Ergebnis des Aufrufes t_datarq(), weil der empfangende Partner den Datenfluß gestoppt hat oder weil in CMX ein zeitweiser Betriebsmittelengpaß vorliegt. Die Daten wurden noch gesendet, aber beim empfangenden Partner nicht mehr angezeigt. Der sendende Partner muß nun mit t_event() auf das Ereignis T_DATAGO warten, um erneut Daten senden zu können.

Soll die CMX-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen Sie anstatt der Aufrufe mit dem Prefix t_ die Aufrufe mit dem Prefix x_ benutzen. Gleiches gilt für die Ereignisse, wo der Prefix T_ durch X_ zu ersetzen ist. Beachten Sie bitte, daß bei ICMX(NEA) auch beim Aufruf x_datain bzw. x_xdatin als Ergebnis der Wert X_DATASTOP gemeldet werden kann. Bevor Sie selbst Daten senden können, müssen Sie dazu auf das Ereignis X_DATAGO warten.

4.4 Beispiel zur Datenübertragung

Die folgenden beiden Programme zeigen den Austausch von Daten. Das Programm data_a ist der sendende Partner, das Programm data_p ist der empfangende Partner. Die Daten werden hin und her geschickt und geprüft, ob sie richtig übertragen werden.

Beispiele für das Senden und Empfangen von Daten mit ICMX(NEA) finden Sie bei der Beschreibung der entsprechenden Aufrufe im Abschnitt 6.3.

```
/*
 * CMX Test Command Source File
 *
 * Aktiver Partner fuer Datenaustauschtest:
 * Der aktiver Partner baut die Verbindung zum passiven Partner auf,
 * sendet ihm eine TSDU und wartet auf deren "Reflexion" durch den
 * passiven Partner. Die empfangene TSDU wird mit der gesendeten
 * verglichen (optional). Danach erfolgen weitere derartigen Zyklen.
 * Der aktive Partner baut die Verbindung aus eigener Initiative bei
 * aufgetretenen Fehlern ab und nach der letzten TSDU.
 * Die Verarbeitung erfolgt TS-Ereignis-gesteuert.
 * Alle Ausgaben erfolgen auf stderr, Fehlermeldungen unbedingt, andere
 * Meldungen abhaengig von -d Option
 *
 * Aufruf:
 *      data_a [-cc] [-d] [-f] [-lmsgsize] [-nmsgcnt] [-omyname] [-pname]
 *              c       Pruefung der TSDU auf Inhalt c
 *              d       debugging, mehrere 'd'  → mehr Informationen
 *                      d       Verbindungsphase
 *                      dd      + Kommunikationsphase
 *                      ddd     + Argumente
 *              f       stderr wird in Datei "ERRxxxxx" mit pid xxxxx geleitet
 *              l       TSDU-Laenge msgsize <= 20000 (default: 20000)
 *              n       TSDU-Anzahl msgcnt (default: 1000)
 *              o       eigener Name myname
 *                      (default: "TestanwendungAKT")
 *              p       Partnername paddr
 *                      (default: "TestanwendungPAS")
 */
static char sccsid[] = "@(#)data_a.c    1.38 87/03/13 (DF154)";

#include        <stdio.h>
#include        <signal.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <sys/types.h>
#include        <sys/timeb.h>

#define FALSE   0
#define TRUE    1
#define ERROR   TRUE
#define OK      FALSE

/*
 * Sende- und Empfangspuffer
 */
#define MAXLNG  20000
#define MAXCNT  1000
char    *e_buf;                 /* TSDU-Puffer fuer Empfang und Senden */

char    *e_bufpt;               /* aktueller Pufferzeiger */
int     e_bufl;                 /* Uebertragungslaenge */
int     e_bufcnt;               /* aktueller Uebertragungszaehler */

char    *s_bufpt;               /* aktueller Pufferzeiger */
int     s_bufl;                 /* Uebertragungslaenge */
int     s_bufcnt;               /* aktueller Uebertragungszaehler */

int     msgsize = MAXLNG;       /* gewuenschte TSDU-Laenge */
int     msgcnt = MAXCNT;        /* Zu uebertragene TSDUs */
char    pattern = '>';          /* Zeichen fuer TSDU */
int     curscnt;                /* laufende TSDU-Nummer f. Senden */
int     currcnt;                /* laufende TSDU-Nummer f. Empfang */
int     badmsg;                 /* fehlerhafte TSDU */

int     datastop;               /* Indicator fuer T_DATASTOP/T_DATAGO */
```

```
int     tidul;                  /* TIDU-Laenge */
int     tref;                   /* Transportreferenz */
int     chain;                  /* TSDU-Indikator fuer t_datarq(), t_datain() */
int     reason;                 /* Grund fuer Verbindungsabbau */
int     pid;                    /* eigene Prozessid */
int     mode;                   /* Modus fuer t_event() */

int     debug = 0;
int     checking = 0;
int     sig;
char    rest[9];                /* Enthaelt die Rechnerzeit in
                                /* <min>:<sek>.<millisek>,    */
                                /* Stunden werden wie Rueckkehr der
                                /* rechzeit() geliefert */

#define SENDEN  1
#define EMPFANG 2
int     state;

/*
 * Struktur der t_info()-Option
 */
struct  t_opti1  option = { T_OPTI1 };

/*
 * Strukturen fuer Adressierung
 */
#define MYNAME  "TestanwendungAKT"
#define PNAME   "TestanwendungPAS"

char myname[TS_LPN+1] = { MYNAME } ;
char pname[TS_LPN+1] = { PNAME } ;
union   t_address myaddress;
union   t_address partaddress;

struct  t_opta1 t_opta1 = { T_OPTA1, T_ACTIVE, 1 };
struct  t_optc1  t_optc1;

extern  int     errno;
int     interrupt();            /* Zum Fangen von SIGINT (DEL) */
char    *malloc();

        /* PARAMETERSTRUKTUREN FUER DIRECTORY SYSTEM */
        /* ========================================= */

              /* Standardkopf */
Ts_head       ts_head = { TS_1VER01 } ;

        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Properties" */
        /* ------------------------------------------------------ */

Ts_leaf_name ts_myname   = { 1, TS_PN, NULL, myname } ;
Ts_leaf_name ts_partname = { 1, TS_PN, NULL, pname } ;

Ts_property ts_prop[] = { NULL, TS_ITEM, NULL, NULL,
                          TS_EMPTYPROP };

Ts_P13 ts1003 = { &ts_head,             /* Standardkopf                    */
                  TS_CMX_DIR_ID,        /* Directory-Identifier            */
                  NULL,                 /* Objektname                      */
                  NULL,                 /* irrelevanter Parameter          */
                  NULL,                 /* Werteliste fuer Eigenschaften   */
                                        /* Laenge der Werteliste           */
                  sizeof (union t_address),
                  ts_prop,              /* Liste der gewuenschten Eigensch. */
```

```
                NULL, NULL, NULL } ; /* irrelevanter Parameter          */

main(argc, argv)

int argc;
char *argv[];
{
        register i;
        char    errfil[20];
        int     error = 0;
        int     file = 0;
        int     event;

        /*
         * Optionsanalyse
         */
        while (--argc && **++argv == '-') {
                switch (*++*argv) {
                case 'c':
                        pattern = *++*argv;
                        checking++;
                        break;
                case 'd':
                        do {
                                debug++;
                        } while (*++*argv) ;
                        break;
                case 'f':
                        file++;
                        break;
                case 'l':
                        msgsize = atoi(++*argv);
                        if (msgsize > MAXLNG) {
                                fprintf(stderr, "msgsize %d zu gross,
                                MAXLNG = %d\n", msgsize, MAXLNG);
                                exit(ERROR);
                        }
                        break;
                case 'n':
                        msgcnt = atoi(++*argv);
                        break;
                case 'o':
                        strcpy(myname, ++*argv);
                        break;
                case 'p':
                        strcpy(pname, ++*argv);
                        break;
                }
        }
        /*
         * Zur Beendigung des Programms wird SIGINT verwendet, in der
         * Behandlung wird die Anwendung abgemeldet (implizite Verbindungs-
         * abbauten). SIGINT soll auch im Hintergrund wirken, daher ignorieren
         * wir die vorherige Signalbehandlung
         */
        signal(SIGINT, interrupt);
        /*
         * Eventuell stderr ummmelden, ungepufferte Ausgabe
         */
        pid = getpid();
        if (file) {
                sprintf(errfil, "ERR%05d", pid);
                freopen(errfil, "w", stderr);
                setbuf(stderr, NULL);
```

```
}
fprintf(stderr, "msgsize %d, msgcnt %d, pattern '%c',
                myname '%s', pname '%s'\n",
                msgsize, msgcnt, pattern, myname, pname);
/*
 * Datenpuffer initialisieren
 */
if ((e_buf = malloc(msgsize)) == NULL) {
        fprintf(stderr, ">>> FEHLER: nicht genug Speicher\n");
        exit(ERROR);
}
e_bufpt = s_bufpt = e_buf;
e_bufcnt = s_bufcnt = 0;
for (i = 0; i < msgsize; i++)
        e_buf[i] = pattern;
/*
 * Aktive Anwendung bei CMX anmelden
 */
ts1003.ts13leaf_name = &ts_myname ;
ts_myname.ts_lf[0].ts_np_length = strlen (myname) ;
ts1003.ts13prval = &myaddress ;
ts_prop[0].ts_name = TS_LNAME ;
if (ts13_read_properties (&ts1003) == TS_ERROR) {
        fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                ts_head.ts_retcode, ts_head.ts_errclass,
                ts_head.ts_errvalue) ;
        exit (ERROR) ;
} else {
        if (ts_prop[0].ts_length == 0) {
                fprintf (stderr, "TS-Fehler (Eigenschaft
                'eigener Name' nicht gefunden)\n") ;
                exit (ERROR) ;
        }
}

if (t_attach(&myaddress, &t_opta1) == T_ERROR) {
        fprintf(stderr, ">>> FEHLER %d bei t_attach\n", t_error());
        exit(ERROR);
}
fprintf(stderr, "%02d:%s   Anwendung '%s' angemeldet, pid %d\n",
        rechzeit(rest), rest, myname, pid);

/*
 * Verbindung aufbauen zum passiven Partner
 */
ts1003.ts13leaf_name = &ts_partname ;
ts_partname.ts_lf[0].ts_np_length = strlen (pname) ;
ts1003.ts13prval = &partaddress ;
ts_prop[0].ts_name = TS_TRANS ;
if (ts13_read_properties (&ts1003) == TS_ERROR) {
        fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                ts_head.ts_retcode, ts_head.ts_errclass,
                ts_head.ts_errvalue) ;
        exit (ERROR) ;
} else {
        if (ts_prop[0].ts_length == 0) {
                fprintf (stderr, "TS-Fehler (Eigenschaft
                'Partneradresse' nicht gefunden)\n") ;
                exit (ERROR) ;
        }
}

if (t_conrq(&tref, &partaddress, &myaddress, NULL) == T_ERROR) {
        fprintf(stderr, ">>> FEHLER %d bei t_conrq, tref 0x%x\n",
                t_error(), tref);
        detach(ERROR);
```

```
}
/*
 * Ereignisgesteuerte Verarbeitung:
 * i.a. t_event() synchron (T_WAIT) wartend
 */
mode = T_WAIT;
for (;;) {
        switch (event = t_event(&tref, mode, NULL)) {

        case T_CONCF:
                /*
                 * Verbindungsaufbau gelungen?
                 */
                if (t_concf(&tref, NULL) == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d bei t_concf
                        tref 0x%x\n",
                                t_error(), tref);
                        detach(ERROR);
                }
                if (t_info(&tref, &option) == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d bei t_info
                        tref 0x%x\n",
                                t_error(), tref);
                        disrq(ERROR);
                }
                tidul = option.t_maxl;
                if (debug) {
                        fprintf(stderr, "%02d:%s   Verbindung zu '%s'
                        aufgebaut, TIDU: %d\n",
                                rechzeit(rest), rest, pname, tidul);
                }
                /*
                 * Senden der 1. TSDU
                 */
                state = SENDEN;
                send();
                break;

        case T_DATAIN:
                /*
                 * Datenankunft vom Partner
                 * TIDU im Puffer speichern
                 */
                e_bufl = tidul;         /* Lesen mit TIDU-Laenge */
                if (t_datain(&tref,e_bufpt,&e_bufl,&chain)
                   == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d bei t_datain
                                tref 0x%x\n", t_error(), tref);
                        disrq(ERROR);
                }
                e_bufpt += e_bufl;
                e_bufcnt += e_bufl;
                if (e_bufcnt > msgsize) {
                        fprintf(stderr, ">>> FEHLER: zu lange TSDU,
                                Soll: %d, Ist: %d\n",
                                msgsize, e_bufcnt);
                        disrq(ERROR);
                }
                if (chain == T_MORE)
                        break;
                if (e_bufcnt < msgsize) {
                        fprintf(stderr, ">>> FEHLER: zu kurze TSDU,
                                Soll: %d, Ist: %d\n",
                                msgsize, e_bufcnt);
                        disrq(ERROR);
                }
```

```
                        /*
                         * TSDU komplett:
                         * Puffer zuruecksetzen, TSDU vergleichen
                         */
                        currcnt++;
                        if (checking)
                                check();
                        if (debug) {
                                fprintf(stderr, "%02d:%s   %3d. von %d TSDU
                                %s\n", rechzeit(rest), rest, currcnt, msgcnt,
                                debug > 1 ? "empfangen" : "");
                        }
                        if (currcnt == msgcnt)
                                disrq(OK);
                        e_bufpt = e_buf;
                        e_bufcnt = 0;
                        state = SENDEN;
                        send();
                        break;

                case T_DATAGO:
                        /*
                         * Nach T_DATASTOP beim Senden. Evtl. weitersenden
                         */
                        datastop = FALSE;
                        send();
                        break;

                case T_ERROR:
                        fprintf(stderr, ">>> FEHLER %d bei t_event\n",
                        t_error()); detach(ERROR);
                        break;

                case T_DISIN:
                        /*
                         * Verbindungsabbau durch Partner oder System
                         */
                        if (t_disin(&tref, &reason, NULL) == T_ERROR) {
                                fprintf(stderr, ">>> FEHLER %d bei
                                        t_disin tref 0x%x\n",
                                        t_error(), tref);
                                detach(ERROR);
                        }
                        if (debug) {
                                fprintf(stderr, "%02d:%s   Verbindungsabbau
                                        erhalten, tref 0x%x, reason %d\n",
                                        rechzeit(rest), rest, tref, reason);
                        }
                        detach(OK);
                        break;

                case T_NOEVENT:
                        /*
                         * Abbruch durch Signal
                         */
                        if (sig)
                                detach(OK);
                        break;

                default:
                        fprintf(stderr, "Unerwarteter event %d\n", event);
                        detach(ERROR);
                }
        }
}
```

```
/*
 * Verbindungsabbau
 */
disrq(result)
{
        if (t_disrq(&tref, NULL) == T_ERROR)
                fprintf(stderr, ">>> FEHLER %d bei t_disrq tref 0x%x\n",
                        t_error(), tref);
        if (debug) {
                fprintf(stderr, "%02d:%s   Verbindung tref 0x%x aktiv
                        abgebaut %s\n", rechzeit(rest), rest, tref,
                        result == ERROR ? "nach Fehler" : "");
        }
        detach(result);
}

/*
 * TSDU an Partner senden
 */
send()
{
        int             rc;

        if (datastop || state == EMPFANG)
                return;
        /*
         * TSDU ggfs. in Stuecke der TIDU-Laenge aufspalten
         * Die TSDU ist in s_buf, s_bufpt zeigt auf das naechste
         * zu sendende Byte der TSDU, s_bufcnt enthaelt die Anzahl
         * der bereits uebertragenen Byte.
         */
        while (s_bufcnt < msgsize) {
                s_bufl = msgsize - s_bufcnt;
                if (s_bufl > tidul) {
                        s_bufl = tidul;
                        chain = T_MORE;
                } else
                        chain = T_END;
                if ((rc = t_datarq(&tref, s_bufpt, &s_bufl, &chain))
                   == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d bei t_datarq
                                tref 0x%x\n", t_error(), tref);
                        disrq(ERROR);
                }
                s_bufpt += s_bufl;
                s_bufcnt += s_bufl;
                if (rc == T_DATASTOP) {
                        datastop = TRUE;
                        break;
                }
        }
        /*
         * Falls TSDU komplett gesendet, Puffer zuruecksetzen
         */
        if (s_bufcnt == msgsize) {
                s_bufpt = e_buf;
                s_bufcnt = 0;
                curscnt++;
                if (debug > 1) {
                        fprintf(stderr, "%02d:%s   %3d. von %d TSDU ge-
                        sendet\n", rechzeit(rest), rest, curscnt, msgcnt);
                }
                state = EMPFANG;
        }
}
```

```
/*
 * TSDU-Vergleich
 * 1. Fehlerhaftes Byte protokollieren bei debug
 * fehlerhafte TSDU zaehlen
 */
check()
{
        register i, j;

        for (i = 0, j = 0; i < msgsize; i++) {
                if (e_buf[i] != pattern) {
                        j++;            /* Fehler aufgetreten */
                        break;
                }
        }
        if (j != 0) {
                j = i + 1;
                fprintf(stderr, ">>> FEHLER in TSDU %d im %d. Zeichen,",
                        currcnt, j);
                fprintf(stderr, "Soll: '%c', Ist: '%c'\n", pattern, e_buf[i]);
                badmsg++;
        }
}

detach(result)
{
        if (t_detach(&myaddress) == T_ERROR)
                fprintf(stderr, ">>> FEHLER %d bei t_detach\n", t_error());
        fprintf(stderr, "%02d:%s   Anwendung '%s', abgemeldet %s, pid %d\n",
                rechzeit(rest), rest, myname,
                result == ERROR ? "wegen Fehler" : "", pid);
        fflush(stderr);
        fclose(stderr);
        exit(result);
}

/*
 * Signalroutine fuer SIGINT
 */
interrupt()
{
        sig++;
}
/* ========================================================================
 * Die Funktion rechnet die absolute Zeit, in der die
 * bestimmte Tat ausgefuehrt ist.
 *  EINGABE: 'wert' Pointer zu einem Feld wo die Zeit gespeichert wird.
 *  AUSGABE: 'hass' Stundewert
 */
rechzeit(wert)
char *wert;
{
        struct timeb pt;
        int hass, mass, dec;

        ftime(&pt);
        if((hass = pt.time/3600%24 - pt.timezone/60) == 24)     /* Stunden */
                hass = 0;
        mass = pt.time%3600/60;         /* Minuten */
        dec = pt.time%3600%60;          /* Sekunden */
        sprintf(wert,"%02d:%02d.%03u", mass, dec, pt.millitm);
        return(hass);
}
```

```
/*
 * CMX Test Command Source File
 *
 * Passiver Partner fuer Datenaustauschtest:
 * Der passive Partner erwartet den Verbindungsaufbau (und -abbau) seitens
 * des aktiven Partners, nimmt TSDUs entgegen, prueft diese und sendet
 * sie sofort zurueck.
 * Der passive Partner baut die Verbindung aus eigener Initiative nur bei
 * aufgetretenen Fehlern ab, sonst erwartet er den Abbau durch den aktiven.
 * Die Verarbeitung erfolgt TS-Ereignis-gesteuert.
 * Alle Ausgabe erfolgen auf stderr, Fehlermeldungen unbedingt, andere
 * Meldungen abhaengig von -d Option
 *
 * Aufruf:
 *      data_p [-cc] [-d] [-f] [-lmsgsize] [-nmsgcnt] [-omyname] [-pname]
 *              c       Pruefung der TSDU auf Inhalt c
 *              d       debugging, mehrere 'd'  -> mehr Informationen
 *                      d       Verbindungsphase
 *                      dd      + Kommunikationsphase
 *                      ddd     + Argumente
 *              f       stderr wird in Datei "ERRxxxxx" mit pid xxxxx geleitet
 *              l       TSDU-Laenge msgsize <= 20000 (default: 20000)
 *              n       TSDU-Anzahl msgcnt (default: 1000)
 *              o       eigener Name myname
 *                      (default: "TestanwendungPAS")
 *              p       Name des Partners pname
 *                      (default: "TestanwendungAKT")
 */
static char sccsid[] = "@(#)data_p.c    1.33 87/01/30 (DF154)";

#include        <stdio.h>
#include        <signal.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <sys/types.h>
#include        <sys/timeb.h>

#define TRUE    1
#define FALSE   0
#define OK      FALSE
#define ERROR   TRUE

/*
 * Sende- und Empfangspuffer
 */
#define MAXLNG  20000
#define MAXCNT  1000

char    *e_buf;                 /* TSDU-Puffer fuer Empfang und Senden */

char    *e_bufpt;               /* aktueller Pufferzeiger */
int     e_bufl;                 /* Uebertragungslaenge */
int     e_bufcnt;               /* aktueller Uebertragungszaehler */

char    *s_bufpt;               /* aktueller Pufferzeiger */
int     s_bufl;                 /* Uebertragungslaenge */
int     s_bufcnt;               /* aktueller Uebertragungszaehler */

int     msgsize = MAXLNG;       /* gewuenschte TSDU-Laenge */
int     msgcnt = MAXCNT;        /* Zu uebertragene TSDUs */
char    pattern = '>';          /* Zeichen fuer TSDU */
int     curscnt;                /* laufende TSDU-Nummer f. Senden */
int     currcnt;                /* laufende TSDU-Nummer f. Empfang */
int     badmsg;                 /* fehlerhafte TSDU */

int     datastop;               /* Indicator fuer T_DATASTOP/T_DATAGO */

int     tidul;                  /* TIDU-Laenge */
```

```
int     tref;                   /* Transportreferenz */
int     chain;                  /* TSDU-Indikator fuer t_datarq(), t_datain() */
int     reason;                 /* Grund fuer Verbindungsabbau */
int     pid;                    /* eigene Prozessid */
int     mode;                   /* Modus fuer t_event() */

int     debug = 0;
int     checking = 0;
int     sig;
char    rest[9];                /* Enthaelt die Rechnerzeit in
                                /* <min>:<sek>.<millisek>,    */
                                /* Stunden werden wie Rueckker der
                                /* rechzeit() geliefert */

#define SENDEN  1
#define EMPFANG 2
int     state;

/*
 * Struktur der t_info()-Option
 */
struct  t_opti1  option = { T_OPTI1 };

/*
 * Strukturen fuer Adressierung
 */
#define PNAME   "TestanwendungAKT"
#define MYNAME  "TestanwendungPAS"

char myname[TS_LPN+1] = { MYNAME } ;
char pname[TS_LPN+1] = { PNAME } ;
union   t_address myaddress;
union   t_address partaddress;

struct  t_opta1 t_opta1 = { T_OPTA1, T_PASSIVE, 1 };

extern  int     errno;
int     interrupt();            /* Zum Fangen von SIGINT (DEL) */
char    *malloc();

        /* PARAMETERSTRUKTUREN FUER DIRECTORY SYSTEM */
        /* ========================================= */

                /* Standardkopf */
Ts_head       ts_head = { TS_1VER01 } ;

        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Leafs" */
        /* ------------------------------------------------ */

Ts_leaf_name ts_sname = { 0 } ;
Ts_leaf_name ts_pname ;
char         ts_nlist[TS_LPN] ;
Ts_filter ts_filter = { TS_TRANS, NULL, (char *) &partaddress } ;

Ts_P11 ts1001 = { &ts_head,         /* Standardkopf                 */
                  TS_CMX_DIR_ID,    /* Directory-Identifier         */
                  &ts_sname,        /* Suchkriterium (Name)         */
                  TS_GENERAL,       /* Suchmodus                    */
                  &ts_filter,       /* Suchkriterium (Eigenschaft)  */
                  TS_LOCAL,         /* Suchmodus                    */
                  NULL, NULL,       /* irrelevante Parameter        */
                  ts_nlist,         /* Werteliste fuer Namensteile  */
                  sizeof(ts_nlist), /* Laenge der Werteliste        */
                  1,                /* Anzahl der erwarteten Namen  */
                  &ts_pname,        /* Liste fuer gefundene Objekte */
                  NULL } ;          /* Anzahl der gefundenen Objekte */
                                    /* (Rueckgabeparameter)         */
```

```
        /* PARAMETERSTRUKTUREN FUER FUNKTION "TS_Read_Properties" */
        /* ------------------------------------------------------ */

Ts_leaf_name ts_myname   = { 1, TS_PN, NULL, myname } ;

Ts_property ts_prop[] = { NULL, TS_ITEM, NULL, NULL,
                          TS_EMPTYPROP };

Ts_P13 ts1003 = { &ts_head,            /* Standardkopf                    */
                  TS_CMX_DIR_ID,       /* Directory-Identifier            */
                  NULL,                /* Objektname                      */
                  NULL,                /* irrelevanter Parameter          */
                  NULL,                /* Werteliste fuer Eigenschaften   */
                                       /* Laenge der Werteliste           */
                  sizeof (union t_address),
                  ts_prop,             /* Liste der gewuenschten Eigensch. */
                  NULL, NULL, NULL } ; /* irrelevanter Parameter          */

/* ------------------------------------------------------------------------ */

main(argc, argv)
int argc;
char *argv[];
{
        register i;
        char    errfil[20];
        int     error = 0;
        int     file = 0;
        int     event;

        /*
         * Optionsanalyse
         */
        while (--argc && **++argv == '-') {
                switch (*++*argv) {
                case 'c':
                        pattern = *++*argv;
                        checking++;
                        break;
                case 'd':
                        do {
                                debug++;
                        } while (*++*argv) ;
                        break;
                case 'f':
                        file++;
                        break;
                case 'l':
                        msgsize = atoi(++*argv);
                        if (msgsize > MAXLNG) {
                                fprintf(stderr, "msgsize %d zu gross,
                                MAXLNG = %d\n",
                                msgsize, MAXLNG);
                                exit(ERROR);
                        }
                        break;
                case 'n':
                        msgcnt = atoi(++*argv);
                        break;
                case 'o':
                        strcpy(myname, ++*argv);
                        break;
                case 'p':
                        strcpy(pname, ++*argv);
                        break;
                }
        }
```

```
/*
 * Zur Beendigung des Programms wird SIGINT verwendet, in der
 * Behandlung wird die Anwendung abgemeldet (implizite Verbindungs-
 * abbauten). SIGINT soll auch im Hintergrund wirken, daher
 * ignorieren wir die vorherige Signalbehandlung
 */
signal(SIGINT, interrupt);
/*
 * Eventuell stderr ummelden, ungepufferte Ausgabe
 */
pid = getpid();
if (file) {
        sprintf(errfil, "ERR%05d", pid);
        freopen(errfil, "w", stderr);
        setbuf(stderr, NULL);
}
fprintf(stderr, "msgsize %d, msgcnt %d, pattern '%c', myname '%s',
                pname '%s'\n", msgsize, msgcnt, pattern, myname,
                pname);
/*
 * Datenpuffer initialisieren
 */
if ((e_buf = malloc(msgsize)) == NULL) {
        fprintf(stderr, ">>> FEHLER: nicht genug Speicher\n");
        exit(ERROR);
}
e_bufpt = s_bufpt = e_buf;
e_bufcnt = s_bufcnt = 0;
/*
 * Passive Anwendung bei CMX anmelden
 */
ts1003.ts13leaf_name = &ts_myname ;
ts_myname.ts_lf[0].ts_np_length = strlen (myname) ;
ts1003.ts13prval = &myaddress ;
ts_prop[0].ts_name = TS_LNAME ;
if (ts13_read_properties( &ts1003 ) == TS_ERROR) {
        fprintf (stderr, "TS-Fehler (Typ: %d Klasse: %d Wert: %d)\n",
                ts_head.ts_retcode, ts_head.ts_errclass,
                ts_head.ts_errvalue) ;
        exit (ERROR) ;
} else {
        if (ts_prop[0].ts_length == 0) {
                fprintf (stderr, "TS-Fehler (Eigenschaft
                'eigener Name' nicht gefunden)\n") ;
                exit (ERROR) ;
        }
}
if (t_attach(&myaddress, &t_opta1) == T_ERROR) {
        fprintf(stderr, ">>> FEHLER %d bei t_attach\n", t_error());
        exit(ERROR);
}
fprintf(stderr, "%02d:%s   Anwendung '%s' angemeldet, pid %d\n",
        rechzeit(rest), rest, myname, pid);

/*
 * Ereignisgesteuerte Verarbeitung:
 * i.a. t_event() synchron (T_WAIT) wartend
 */
mode = T_WAIT;
for (;;) {
        switch (event = t_event(&tref, mode, NULL)) {

        case T_CONIN:
                /*
                 * Verbindungsaufbauwunsch akzeptieren
                 */
                if (t_conin(&tref, &myaddress, &partaddress, NULL)
                        == T_ERROR) {
```

```
                fprintf(stderr, ">>> FEHLER %d bei t_conin
                        tref 0x%x\n", t_error(), tref);
                detach(ERROR);
        }
        if (t_conrs(&tref, NULL) == T_ERROR) {
                fprintf(stderr, ">>> FEHLER %d bei t_conrs
                        tref 0x%x\n", t_error(), tref);
                detach(ERROR);
        }
        if (t_info(&tref, &option) == T_ERROR) {
                fprintf(stderr, ">>> FEHLER %d bei t_info
                        tref 0x%x\n", t_error(), tref);
                disrq(ERROR);
        }
        tidul = option.t_maxl;
        if (debug) {
                ts_filter.ts_pr_length = partaddress.
                tpartaddr.t_palng ;
                if (ts11_read_leafs (&ts1001) == TS_ERROR) {
                        fprintf (stderr, "TS-Fehler (Typ:
                        %d Klasse: %d Wert: %d)\n",
                                ts_head.ts_retcode, ts_head.
                                ts_errclass, ts_head.ts_errvalue) ;
                        exit (ERROR) ;

                }
                ts_nlist[ts_pname.ts_lf[0].ts_np_length] =
                '\0' ;
                fprintf(stderr, "%02d:%s   Verbindung zu '%s'
                aufgebaut, TIDU: %d\n",
                        rechzeit(rest), rest, ts_nlist, tidul);
        }
        break;

case T_DATAIN:
        /*
         * Datenankunft vom Partner
         * TIDU im Puffer speichern
         */
        e_bufl = tidul;         /* Lesen mit TIDU-Laenge */
        if (t_datain(&tref,e_bufpt,&e_bufl,&chain)
        == T_ERROR) {
                fprintf(stderr, ">>> FEHLER %d bei
                       t_datain tref 0x%x\n",
                        t_error(), tref);
                disrq(ERROR);
        }
        e_bufpt += e_bufl;
        e_bufcnt += e_bufl;
        if (e_bufcnt > msgsize) {
                fprintf(stderr, ">>> FEHLER: zu lange TSDU,
                        Soll: %d, Ist: %d\n",
                        msgsize, e_bufcnt);
                disrq(ERROR);
        }
        if (chain == T_MORE)
                break;
        if (e_bufcnt < msgsize) {
                fprintf(stderr, ">>> FEHLER: zu kurze TSDU,
                        Soll: %d, Ist: %d\n",
                        msgsize, e_bufcnt);
                disrq(ERROR);
        }
        /*
         * TSDU komplett:
         * Puffer zuruecksetzen, TSDU vergleichen
         */
        currcnt++;
```

```
                        if (checking)
                                check();
                        if (debug > 1) {
                                fprintf(stderr, "%02d:%s   %3d. von %d TSDU
                                        empfangen\n", rechzeit(rest), rest,
                                        currcnt, msgcnt);
                        }
                        e_bufpt = e_buf;
                        e_bufcnt = 0;
                        state = SENDEN;
                        send();
                        break;

                case T_DATAGO:
                        /*
                         * Nach T_DATASTOP beim Senden. Evtl. weitersenden
                         */
                        datastop = FALSE;
                        send();
                        break;

                case T_ERROR:
                        fprintf(stderr, ">>> FEHLER %d bei t_event\n",
                        t_error()); detach(ERROR);
                        break;

                case T_DISIN:
                        /*
                         * Verbindungsabbau durch Partner oder System
                         */
                        if (t_disin(&tref, &reason, NULL) == T_ERROR) {
                                fprintf(stderr, ">>> FEHLER %d bei t_disin
                                        tref 0x%x\n", t_error(), tref);
                                detach(ERROR);
                        }
                        if (debug) {
                                fprintf(stderr, "%02d:%s   Verbindungsabbau
                                        erhalten, tref 0x%x, reason %d\n",
                                        rechzeit(rest), rest, tref, reason);
                        }
                        detach(OK);
                        break;

                case T_NOEVENT:
                        /*
                         * Abbruch durch Signal
                         */
                        if (sig)
                                detach(OK);
                        break;

                default:
                        fprintf(stderr, "Unerwarteter event %d\n", event);
                        detach(ERROR);
                }
        }
}

/*
 * Verbindungsabbau
 */
disrq(result)
{
        if (t_disrq(&tref, NULL) == T_ERROR)
                fprintf(stderr, ">>> FEHLER %d bei t_disrq tref 0x%x\n",
                        t_error(), tref);
        if (debug) {
                fprintf(stderr, "%02d:%s   Verbindung tref 0x%x aktiv
```

```
                        abgebaut %s\n",
                        rechzeit(rest), rest, tref,
                        result == ERROR ? "nach Fehler" : "");
        }
        detach(result);
}

/*
 * TSDU an Partner senden
 */
send()
{
        int     rc;

        if (datastop || state == EMPFANG)
                return;
        /*
         * TSDU ggfs. in Stuecke der TIDU-Laenge aufspalten
         * Die TSDU ist in s_buf, s_bufpt zeigt auf das naechste
         * zu sendende Byte der TSDU, s_bufcnt enthaelt die Anzahl
         * der bereits uebertragenen Byte.
         */
        while (s_bufcnt < msgsize) {
                s_bufl = msgsize - s_bufcnt;
                if (s_bufl > tidul) {
                        s_bufl = tidul;
                        chain = T_MORE;
                } else
                        chain = T_END;
                if ((rc = t_datarq(&tref, s_bufpt, &s_bufl, &chain))
                        == T_ERROR) {
                        fprintf(stderr, ">>> FEHLER %d bei t_datarq
                                tref 0x%x\n", t_error(), tref);
                        disrq(ERROR);
                }
                s_bufpt += s_bufl;
                s_bufcnt += s_bufl;
                if (rc == T_DATASTOP) {
                        datastop = TRUE;
                        break;
                }
        }
        /*
         * Falls TSDU komplett gesendet, Puffer zuruecksetzen
         */
        if (s_bufcnt == msgsize) {
                s_bufpt = e_buf;
                s_bufcnt = 0;
                curscnt++;
                if (debug) {
                        fprintf(stderr, "%02d:%s   %3d. von %d TSDU %s\n",
                                rechzeit(rest), rest, currcnt, msgcnt,
                                debug > 1 ? "gesendet" : "");
                }
                state = EMPFANG;
        }
}

/*
 * TSDU-Vergleich
 * 1. Fehlerhaftes Byte protokollieren bei debug
 * fehlerhafte TSDU zaehlen
 */
check()
{
        register i, j;

        for (i = 0, j = 0; i < msgsize; i++) {
```

```
                if (e_buf[i] != pattern) {
                        j++;            /* Fehler aufgetreten */
                        break;
                }
        }
        if (j != 0) {
                j = i + 1;
                fprintf(stderr, ">>> FEHLER in TSDU %d im %d. Zeichen,",
                        currcnt, j);
                fprintf(stderr, "Soll: '%c', Ist: '%c'\n", pattern,
                e_buf[i]); badmsg++;
        }
}

detach(result)
{
        if (t_detach(&myaddress) == T_ERROR)
        fprintf(stderr, "%02d:%s   Anwendung '%s' abgemeldet %s, pid %d\n",
                rechzeit(rest), rest, myname,
                result == ERROR ? "wegen Fehler" : "", pid);
        fflush(stderr);
        fclose(stderr);
        exit(result);
}

/*
 * Signalroutine fuer SIGINT
 */
interrupt()
{
        sig++;
}
/* ========================================================================
 * Die Funktion rechnet die absolute Zeit, in der die
 * bestimmte Tat ausgefuehrt ist.
 *  EINGABE: 'wert' Pointer zu einem Feld wo der Zeit gespeichert wird.
 *  AUSGABE: 'hass' Stundewert
 */
rechzeit(wert)
char *wert;
{
        struct timeb pt;
        int hass, mass, dec;

        ftime(&pt);
        if((hass = pt.time/3600%24 - pt.timezone/60) == 24)     /* Stunden */
                hass = 0;
        mass = pt.time%3600/60;         /* Minuten */
        dec = pt.time%3600%60;          /* Sekunden */
        sprintf(wert,"%02d:%02d.%03u", mass, dec, pt.millitm);
        return(hass);
}
```

5 Wie man den Transport Name Service (TNS) verwendet

Die Funktionen des TNS ermöglichen es, konfigurationsspezifische Eigenschaften von Kommunikationspartnern, wie TRANSPORTADRESSE und LOKALEN NAMEN, in einer Datenbasis, dem TS-Directory, zu verwalten. Eine Übersicht über die Funktionen enthält Abschnitt 1.2.1.

Der TNS bietet hierfür eine benutzerfreundliche Namensgebung, mit der man sich auf Kommunikationspartner mit Namen beziehen kann, die insbesondere für die Verwendung durch den Menschen geeignet sind. Er übernimmt die erforderliche Abbildung GLOBALEN NAMEN < > Eigenschaften, die die dynamische Zuordnung von Eigenschaften zu Kommunikationspartnern erlaubt.

Der TNS stellt dem Benutzer auf Anforderung folgendes zur Verfügung:

- Zuordnung von GLOBALEN NAMEN zu Eigenschaften:
  damit wird eine Beziehung zwischen einem GLOBALEN NAMEN und einer Menge von Informationen über einen Kommunikationspartner, auf den sich der GLOBALE NAME bezieht, hergestellt. Dies steht in Analogie zu den "weißen Seiten" des Telefonbuches, das den Namen von Personen oder Organisationen (GLOBALEN NAMEN) als Information (Eigenschaft) die Telefonnummer zuordnet.

- Zuordnung von Eigenschaft zu GLOBALEN NAMEN:
  damit können die GLOBALEN NAMEN von solchen Kommunikationspartnern aufgelistet werden, die eine angegebene Eigenschaft besitzen. Dies steht in Analogie zu den "gelben Seiten" z.B. des Telefonbuches, das dazu verwendet werden kann, alle Klempner einer Stadt zu ermitteln.

Die TNS-Funktionen gruppieren sich in solche, mit denen man Informationen abfragen kann, und in solche, mit denen man die Informationen verändern kann. Letztere verwendet das Verwaltungsprogramm TNSADMIN [CCP], mit dem Sie interaktiv und menügesteuert die Informationen verwalten können.

Die folgenden Abschnitte erläutern die wesentlichen Datenstrukturen und Begriffe, bevor in den weiteren Kapiteln die TNS-Funktionen ausführlich besprochen werden.

Einheitliche Namensstruktur

Die Kommunikationspartner sind durch ihren GLOBALEN NAMEN bezeichnet. Das ist ein hierarchisch strukturierter Name. Er besteht aus einer beliebigen Teilmenge von bis zu 5 Namensteilen, Namensteil1 bis Namensteil5. Von diesen ist Namensteil1 in der Hierarchie der höchste, Namensteil5 der niedrigste. In einem GLOBALEN NAMEN müssen nicht alle Hierarchiestufen vorhanden sein. Der TNS macht, abgesehen von der hierarchischen Reihenfolge, keine weiteren Vorgaben hinsichtlich Bedeutung der Namensteile innerhalb des GLOBALEN NAMENS. Aus der hierarchischen Struktur folgt, daß sich die GLOBALEN NAMEN in einem Namensbaum (dem "global naming tree") anordnen lassen. Ein Beispiel eines Namensbaumes zeigt das Bild 5-1. Der Baum besteht aus drei Elementen:
der Wurzel (ROOT),
den Knoten, im folgenden NonLeafEntities genannt,
den Blätter, im folgenden LeafEntities genannt.

Der Baum wächst von oben nach unten.

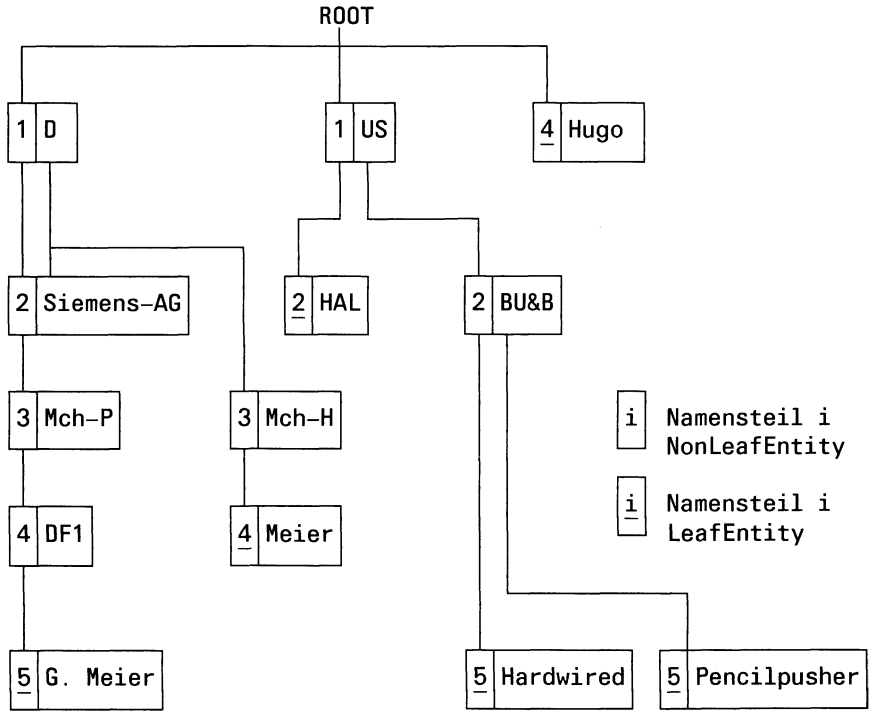


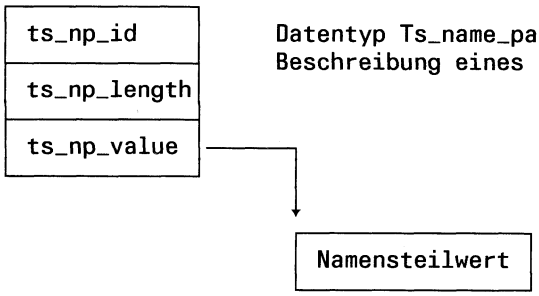
Bild 5-1 Beispiel eines Namensbaumes GLOBALER NAMEN

Es gelten dann folgende Entsprechungen und Merkmale:

- Ein GLOBALER NAME entspricht einem Pfad im Namensbaum von der Wurzel zu einer LeafEntity oder einer NonLeafEntity. Dabei entsprechen die Namensteile den Pfadkomponenten.
- An die Wurzel oder eine NonLeafEntity kann unter Beachtung der Hierarchie ein weiterer Namensteil angehängt und dabei festgelegt werden, ob der resultierende Pfad dann einer NonLeafEntity oder einer LeafEntity entsprechen soll. Ein Namensteil führt also von der Wurzel oder einer NonLeafEntity zu einer anderen NonLeafEntity oder einer LeafEntity.
- Alle Namensteile können Pfadkomponente zu einer LeafEntity sein, außer Namensteil5 können alle Namensteile Pfadkomponente zu einer NonLeafEntity sein.
- Eigenschaften können nur einer LeafEntity zugeordnet werden.

  Ein Namensteil wird durch Identifikation, Länge und Wert beschrieben wie im folgenden Datentyp **Ts_name_part** (siehe tnsx.h):

```
typedef struct {
    long  ts_np_id;          /* Identifikation */
    short ts_np_length;      /* Länge */
    char  *ts_np_value;      /* Wert */
} Ts_name_part;
```



ts_np_id enthält die Identifikation des Namensteils, ts_np_length seine Länge in Oktetts. ts_np_value ist ein Zeiger zu einem Feld, in dem der Wert in der Länge ts_np_length enthalten ist. Die folgende Tabelle enthält die möglichen Werte:

Namensteil	ts_np_id	ts_np_length (dez*)	Bedeutung **)
1	TS_COUNTRY	TS_LCOUNTRY (2)	Country
2	TS_ADMD	TS_LADMD (16)	Administration Domain
3	TS_PRMD	TS_LPRMD (16)	Private Domain
4	TS_OU	TS_LOU (10)	Organizational Unit
5	TS_PN	TS_LPN (30)	Personal Name

*) gegenwärtige Maximallänge gemäß tnsx.h
**) gemäß internationaler Vorschläge

Tabelle 5-1 Identifikationen, Längen und Werte von Namensteilen

Der Wert eines Namensteils darf nur aus abdruckbaren Zeichen des ISO-7-Bit-Codes bestehen.

Ein GLOBALER NAME einer LeafEntity im Namensbaum besteht aus bis zu TS_NO_NP Namensteilen, der einer NonLeafEntity besteht aus bis zu TS_NO_NP - 1 Namensteilen. Beispiele von GLOBALEN NAMEN sind:

		Namensteil		
1	2	3	4	5
D 49	Siemens AG D ST TZ 089	Mch-P Reg18 636	Abteilung1 Proz. 1	NN, Tel. 123-45678 $DIALOG 47658

Der GLOBALE NAME einer NonLeafEntity wird durch den Datentypen **Ts_non_leaf_name**

```
typedef struct {
    short        ts_no_of_el;           /* Anzahl Namensteile */
    Ts_name_part ts_nlf[TS_NO_NP - 1];  /* Namensteile */
} Ts_non_leaf_name;
```

beschrieben, der einer LeafEntity durch den Datentypen **TS_leaf_name**

```
typedef struct {
    short        ts_no_of_el;           /* Anzahl Namensteile */
    Ts_name_part ts_lf[TS_NO_NP];       /* Namensteile */
} Ts_leaf_name;
```

ts_no_of_el enthält in beiden Datentypen die tatsächliche Anzahl der Namensteile. Der TNS erwartet, daß die Namensteile in absteigender Hierarchie eingetragen werden. Das heißt: ts_lf[0] bzw. ts_nlf[0] enthalten den in der Hierarchie höchsten Namensteil, ts_lf[1] bzw. ts_nlf[1] enthalten den nächstniedrigeren usw..

Beispiel

Ein Beispiel für die Deklaration und Versorgung eines Datentypen Ts_leaf_name zur Compilezeit ist:

```
#include <tnsx.h>

Ts_leaf_name    leaf = {
    5,                              /* ts_no_of_el */
    TS_COUNTRY, 1, "D",             /* ts_lf[0] */
    TS_ADMD,   10, "Siemens-AG",    /* ts_lf[1] */
    TS_PRMD,    5, "Mch-P",         /* ts_lf[2] */
    TS_OU,      3, "DF1",           /* ts_lf[3] */
    TS_PN,      8, "G._Meier"       /* ts_lf[4] */
};
```

Bild 5-2 Statische Versorgung von Ts_leaf_name

Da zur Compilezeit die Namensteile nicht immer bekannt sind, zeigt das folgende Programmfragment die Versorgung des Datentypen Ts_leaf_name zur Laufzeit. Die Namensteile werden als Strings in C-Notation im Zeigerfeld "npart[]" mit abfallender Hierarchie erwartet. Die auszufüllende Datenstruktur sei "leaf".

```
#include <tnsx.h>

char          *npart[] = {
    "D",                    /* Namensteil 1 */
    "Siemens-AG",           /* Namensteil 2 */
    "Mch-P",                /* Namensteil 3 */
    "Abteilung1",           /* Namensteil 4 */
    "NN, Tel. 123-45678"    /* Namensteil 5 */
};
Ts_leaf_name leaf;
long         id = TS_COUNTRY;
Ts_name_part *nptr;

    nptr = leaf.ts_lf;
    leaf.ts_no_of_el = 0;
    for (i = 0; i < TS_NO_NP; i++) {
        if ((lng = strlen (npart[i])) != 0) {
            nptr->ts_np_id = id + (long) i;
            nptr->ts_np_length = lng;
            nptr->ts_np_value = npart[i];
            leaf.ts_no_of_el++;
            nptr++;
        }
    }
```

Bild 5-3 Dynamische Versorgung von Ts_leaf_name

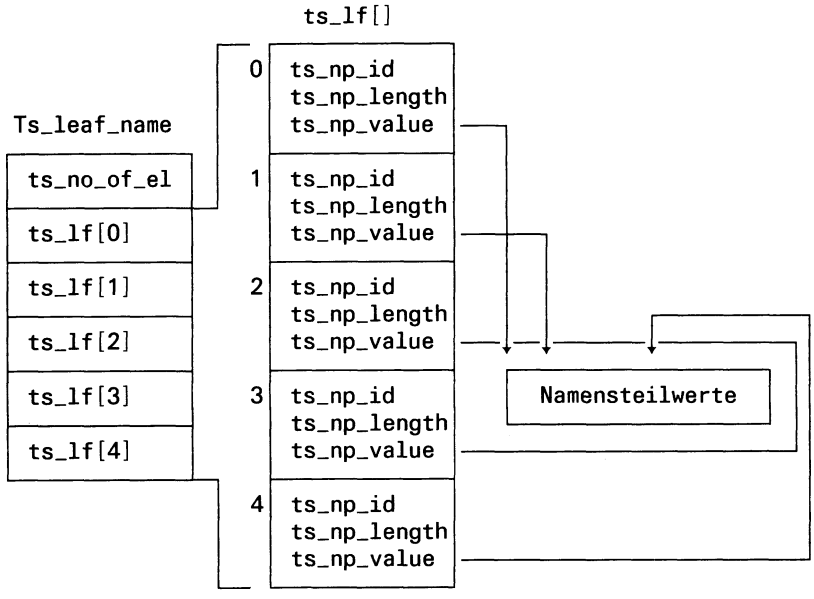


Bild 5-4 Schematische Darstellung von Ts_leaf_name

Eigenschaften

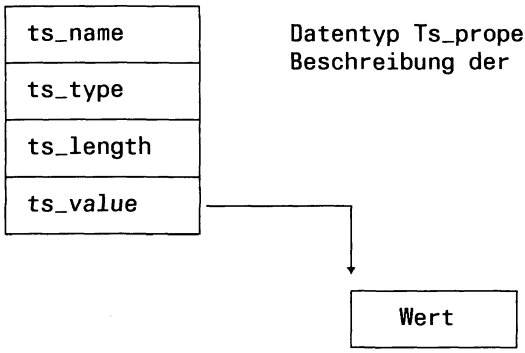
Den LeafEntities im Namensbaum kann eine Auswahl aus 6 Eigenschaften zugeordnet werden. Diese wurden bereits kurz in Abschnitt 2.1 betrachtet. Die folgende Tabelle stellt sie noch einmal zusammen.

Eigenschaft	Bedeutung
TS_LNAME	LOKALER NAME einer lokalen CMX-Anwendung, mit dem sich ihre Prozesse bei CMX anmelden.
TS_TRANS	TRANSPORTADRESSE eines Kommunikationspartners, mit der er im Netz erreichbar ist.
TS_ROUT	ROUTINGINFORMATION bei Mehrfachanschluß an ein und dasselbe oder verschiedene Netzwerke.
TS_NEABX	MIGRATIONSSERVICE gibt an, ob die lokale oder entfernte CMX-Anwendung für die Kommunikation den Migrationsdienst NEABX (also die x_...-Aufrufe von ICMX(NEA)) verwenden muß
TS_GTYPE	GERAETETYP dient zur internen Unterscheidung einer gewöhnlichen von einer Emulationsanwendung.
TS_TRSYS	TRANSPORTSYSTEM klassifiziert die Art des Netzwerkes, über das ein Kommunikationspartner erreichbar ist.

Tabelle 5-2 Übersicht über die in TNS definierten Eigenschaften von TS-Anwendungen

Eine Eigenschaft wird analog einem Namensteil durch eine Identifikation, eine Länge und einen Wert festgelegt. Zusätzlich gibt es noch den Typ der Eigenschaft. Der TNS selbst macht keine Vorgaben hinsichtlich interner Struktur oder Bedeutung der Eigenschaften. Der Datentyp **Ts_property** beschreibt eine Eigenschaft wie folgt:

```
typedef struct {
    short ts_name;          /* Identifikation */
    short ts_type;          /* Typ TS_ITEM/TS_GROUP */
    short ts_length;        /* Länge */
    char  *ts_value;        /* Zeiger auf Wert */
} Ts_property;
```



Datentyp Ts_property zur Beschreibung der Eigenschaften.

ts_name enthält die Identifikation der Eigenschaft, ts_type den Typ TS_GROUP oder TS_ITEM (in Version 1 wird nur TS_ITEM unterstützt!), ts_length die Länge des Wertes in Oktetts. ts_value ist ein Zeiger auf einen Datenbereich mit dem Wert in der Länge ts_length. Der Wert kann aus beliebigen Oktetts bestehen.

Der Datentyp **Ts_filter** beschreibt eine Eigenschaft, falls sie zur Ausfilterung von GLOBALEN NAMEN aus dem TS-Directory verwendet wird.

```
typedef struct {
    short ts_pr_name;       /* Identifikation */
    short ts_pr_length;     /* Länge */
    char  *ts_pr_value;     /* Zeiger auf Wert */
} Ts_filter;
```

ts_pr_name, ts_pr_length und ts_pr_value haben dieselbe Bedeutung wie die entsprechenden Felder in Ts_property. Die folgende Tabelle enthält die möglichen Werte:

Eigenschaft	ts_name ts_pr_name	ts_length ts_pr_length	Inhalt von ts_value und ts_pr_value
LOKALER NAME	TS_LNAME	max. TS_LPROP	beliebig
TRANSPORTADRESSE	TS_TRANS	max. TS_LPROP	beliebig
ROUTINGINFORMATION	TS_ROUT	sizeof(short)	0...16
MIGRATIONSSERVICE	TS_NEABX	sizeof(char)	TS_TRUE oder TS_FALSE
GERAETETYP	TS_GTYPE	sizeof(char)	TS_SS8110, TS_DSS9750 oder TS_DRS8122
TRANSPORTSYSTEM	TS_TRSYS	sizeof(char)	TS_NEA, TS_ISO, TS_STA, oder TS_LAN
leere Eigenschaft	TS_EMPTYPROP	undefiniert	undefiniert

Tabelle 5-3 Mögliche Werte zur Beschreibung einer Eigenschaft

Mehrere Eigenschaften sind als Feld von Datentypen Ts_property angebbar. Eine Reihenfolge ist dabei nicht vorgeschrieben. Das Feld muß aber mit einem Eintrag mit der Identifikation TS_EMPTYPROP in ts_name abgeschlossen werden.

Beispiel

Das folgende Programmfragment zeigt, wie ein Feld von 2 Datentypen Ts_property versorgt wird. Die beiden Eigenschaften TRANSPORT-ADRESSE und ROUTINGINFORMATION mögen in entsprechenden Feldern "trans" und "rout" vorliegen (für deren Struktur vgl. cmx.h):

```
#include <cmx.h>
#include <tnsx.h>

/*
 * Wertefeld für Eigenschaften
 * Feld für Beschreibungen der Eigenschaften
 */
union t_address trans;
Ts_route        rout;

Ts_property prop[] = {
    TS_TRANS, TS_ITEM, NULL, NULL,  /* TRANSPORTADRESSE */
    TS_ROUT,  TS_ITEM, NULL, NULL,  /* ROUTINGINFORM. */
    TS_EMPTYPROP
 };

    /*
     * Längen und Werte der Eigenschaften versorgen
     */
    prop[0].ts_value = (char *) &trans;
    prop[0].ts_length = trans.tpartaddr.t_palng;
    prop[1].ts_value = (char *) &rout;
    prop[1].ts_length = sizeof(rout);
```

Bild 5-5 Versorgung von Ts_property

Die Versorgung der Datenstruktur Ts_filter erfolgt in völlig analoger Weise. Das folgende Bild zeigt die Datenstruktur Ts_property schematisch:

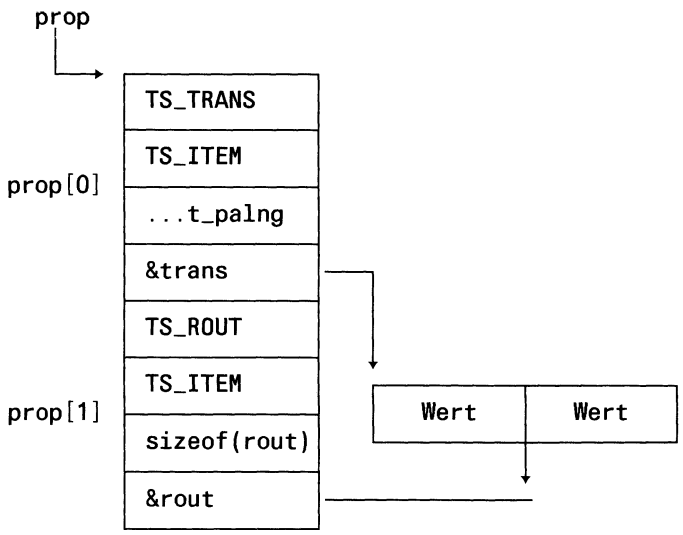


Bild 5-6 Schematische Darstellung von Ts_property

Das TS-Directory

Das TS-Directory besteht aus Einträgen, von denen jeder Informationen über einen Kommunikationspartner in Form von Eigenschaften enthält. Die Einträge sind durch den GLOBALEN NAMEN der Kommunikationspartner bezeichnet und damit in Form des Namensbaumes angeordnet.

Der TNS unterstützt gleichzeitig bis zu 10 verschiedene TS-Directories mit den Identifikationen 0-9. Bei jeder TNS-Funktion muß als Parameter die Identifikation des TS-Directory angegeben werden, auf das sich die Funktion beziehen soll.

Achtung

CMX greift intern auf das TS-Directory TS_CMX_DIR_ID zu. Die CMX-Anwendung muß dies beachten! Sie muß also Eigenschaften, die an CMX zu übergeben sind, im TS-Directory TS_CMX_DIR_ID verwalten.

Die Identifikation des TS-Directory kann bei jedem Aufruf eine andere sein. Häufiges Umschalten zwischen den TS-Directories senkt aber die Performance des TNS, da in solchen Fällen die zur Zugriffsoptimierung verwendeten Caches oft neu aufgebaut werden müssen. Die TS-Directories sind im SINIX-Dateisystem als Dateiverzeichnisse

/usr/lib/cmx/DIR<id> <id> = 0,...,9

abgelegt. Es ist möglich, dieses Dateiverzeichnis mit den üblichen SINIX-Mitteln [SIN1] aufzulisten, die eingetragenen Dateien enthalten aber überwiegend nicht abdruckbare Information.

Achtung

Ein TS-Directory darf nicht mit den üblichen SINIX-Mitteln [SIN1] gelöscht oder auf einen anderen Rechner transportiert werden! Zu diesem Zweck müssen die Funktionen

- ABSPEICHERN eines TS-Directory
- EINLESEN eines TS-Directory
- LOESCHEN eines TS-Directory

im Verwaltungsprogramm TNSADMIN [CCP] verwendet werden. Nur diese stellen zuverlässig sicher, daß die Caches mit den Dateien auf dem Sekundärspeicher richtig synchronisiert werden.

Standardkopf

Allen TNS-Funktionen gemeinsam ist der Datentyp **Ts_head**. Er beschreibt den Standardkopf.

```
typedef struct {
    short ts_version;     /* Versionsnummer */
    short ts_retcode;     /* Meldungstyp */
    short ts_errclass;    /* Meldungsklasse */
    short ts_errvalue;    /* Meldungswert */
} Ts_head;
```

In ts_version ist vor dem Aufruf einer TNS-Funktion die Versionsnummer einzutragen. Das ist die Konstante TS_1VER01 für die Informationsabfrage und TS_2VER01 für die Informationsverwaltung.

In ts_retcode, ts_errclass und ts_errvalue liefert das TNS nach dem Aufruf differenzierte Diagnoseinformation. Diese sind in Abschnitt 6 genauer erläutert.

5.1 Informationen vom TNS abfragen

Folgende Funktionen stellt der TNS zur Informationsabfrage zur Verfügung:

Funktionsaufrufe zur Informationsabfrage	Interpretation
ts11_read_leafs	Auflisten GLOBALER NAMEN mit vorgegebenen Namensteilen und Eigenschaften
ts12_read_children	Auflisten von Namensteilen der nächstniedrigeren Hierarchiestufe
ts13_read_properties	Abfrage der Eigenschaften zu einem GLOBALEN NAMEN

5.1.1 Anwendungen im TS-Directory suchen

Die Suche nach (einem) GLOBALEN NAMEN mit bestimmten Vorgaben im TS-Directory erfolgt durch **ts11_read_leafs()**. Die Vorgaben heißen "Filterkriterien", denn diese Funktion "filtert" aus der Gesamtheit aller GLOBALER NAMEN solche aus, die die Vorgaben erfüllen. Es gibt zwei Filterkriterien:

- vorgegebene Namensteile:
  die der GLOBALE NAME haben muß, um den Filter bei der Suche zu passieren. Sie werden im Parameter ts11partial_name, der vom Typ Ts_leaf_name ist, beschrieben. Dort sind sie in **absteigender** Hierarchie anzugeben. Wie diese Datenstruktur ausgefüllt werden muß, wurde bereits eingangs von Abschnitt 5 bei der "Einheitlichen Namensstruktur" beschrieben (siehe auch Bild 5-3 und 5-4). Soll nicht nach Namensteilen gefiltert werden, so ist einfach in ts_no_of_el von ts11partial_name der Wert 0 einzutragen oder der Parameter ts11 partial-name auf NULL zu setzen.

  Beliebige der angegebenen Namensteile dürfen als Wert genau einen '*' haben. Das bedeutet, die gesuchten GLOBALEN NAMEN müssen den entsprechenden Namensteil besitzen, er darf aber einen beliebigen Wert haben.

Diese Filterung kann weiter eingeschränkt werden durch die Zusatzangabe TS_RESTRICTED oder TS_GENERAL in Parameter ts11str_def. TS_RESTRICTED beschränkt die Filterung nach den GLOBALEN NAMEN auf solche, die genau die vorgegebenen Namensteile haben (evtl. mit beliebigem Wert gemäß '*'). TS_GENERAL filtert dagegen auch solche GLOBALEN NAMEN aus, die neben den vorgegebenen noch weitere, in der Namenshierarchie tieferliegende Namensteile enthalten.

Beispiel

Bei Anwendung des Filterkriteriums

```
Ts_leaf_name nameparts = { 2, TS_COUNTRY, 1, "D", TS_PRMD, 1, "*" };
```

auf den Namensbaum aus Bild 5-1 würde im Fall TS_RESTRICTED kein GLOBALER NAME gefunden (es gibt keinen, der nur den Namensteil TS_PRMD besitzt). Im Fall TS_GENERAL würden die GLOBALEN NAMEN

1	2	3	4	5
D		Mch-H	Meier	

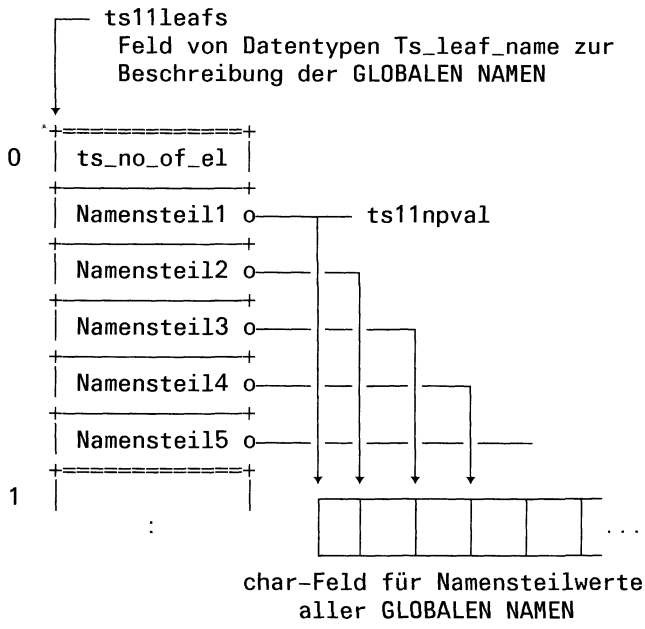
gefunden.

- die Vorgabe einer Eigenschaft eines GLOBALEN NAMENS: Oft soll zu einer vorgegebenen Eigenschaft der GLOBALE NAME ermittelt werden. Z.B. wenn nach einer Verbindungsaufbauanzeige mit t_conin() die TRANSPORTADRESSE des rufenden Partners von CMX erhalten wurde und nun der GLOBALE NAME des rufenden Partners ermittelt werden soll.

  Die vorgegebene Eigenschaft wird durch den Datentyp Ts_filter im Parameter ts11filter beschrieben. Soll nicht nach einer vorgegebenen Eigenschaft gefiltert werden, so ist einfach in ts_pr_name von ts11filter der Wert TS_EMPTYPROP anzugeben. Ist ts_pr_length in ts11filter 0, so können nur die GLOBALEN NAMEN den Filter passieren, die diese Eigenschaft besitzen, wobei sie dann aber einen beliebigen Wert haben kann.

Dem Parameter ts11exp_no entnimmt der TNS, wieviele GLOBALE NAMEN erwartet werden und liefert die gefundenen GLOBALEN NAMEN wahlweise in einer Datenstruktur im Hauptspeicher (Parameter ts11npval, ts111_np_values und ts11leafs) oder in der im Parameter ts11filename angegebenen Datei zurück. Die Anzahl der gefundenen GLOBALEN NAMEN steht nach dem Aufruf in ts11real no. Ist keines der Filterkriterien angegeben, versucht der TNS alle im TS-Directory enthaltenen GLOBALEN NAMEN zurückzuliefern. Dabei kann es bei Ausgabe in den Hauptspeicher ab ca. 50 GLOBALEN NAMEN zu Betriebsmittelengpässen kommen, die zu Funktionsabbruch führen. Diese Engpässe treten bei Ausgabe in eine Datei niemals auf.

Bei Übergabe der GLOBALEN NAMEN im Hauptspeicher muß vor dem Aufruf ausreichend Speicherbereich zur Verfügung stehen für ein Feld von so vielen Datentypen Ts_leaf_name, wie GLOBALE NAMEN erwartet werden (ts11exp_no). Ein Zeiger auf dieses Feld ist im Parameter ts11leafs zu übergeben. Jeder Eintrag in diesem Feld nimmt die Beschreibung eines vom TNS zu liefernden GLOBALEN NAMENs auf, nämlich Anzahl der Namensteile und Beschreibung der Namensteile mit Identfikation, Länge und Zeiger auf dessen Wert. Für die Werte aller Namensteile aller erwarteten GLOBALEN NAMEN erwartet der TNS im Parameter ts11npval einen Zeiger auf ein char-Feld hinreichender Länge, die ihrerseits in ts111_np_values anzugeben ist. In das char-Feld trägt der TNS für jeden Namensteil dessen Wert ein und schreibt Zeiger darauf in den Eintrag des entsprechenden Namensteils unter ts11leafs.



Die Anzahl ts11exp_no der in ts11leafs bereitgestellten Feldelemente für die GLOABLEN NAMEN und die Länge ts11l_np_values des char-Feldes in ts11npval verwendet der TNS zur Kontrolle gegen Überschreibung. Eine Formel zur Berechnung der Länge **ts11l_np_values** des erforderlichen char-Feldes für **ts11exppno** GLOBALE NAMEN ist

```
ts11l_np_values =
   ts11exp_no * (TS_LCOUNTRY + TS_LADMD + TS_LPRMD + TS_LOU + TS_LPN)
```

falls im GLOBALEN NAMEN alle Namensteile mit maximaler Länge auftreten können.

Bei Ausgabe in eine Datei sind die Parameter für die Beschreibung des Hauptspeicherbereiches (ts11exp_no, ts11leafs, ts11npval, ts11l_np_values) nicht relevant, es ist NULL anzugeben. Bei der Angabe des Dateinamens in ts11filename und den Besitz- und Zugriffsrechten der Datei gelten die üblichen SINIX-Konventionen, der Pfad kann relativ oder absolut sein, alle Dateiverzeichnisse des Pfades müssen vorhanden sein. Bei Angabe eines relativen Pfadnamens erfolgt die Ablage der Datei unter "/usr/admin". Das Format der Ausgabe ist datenorientiert, d.h. die Information ist in ein bestimmtes Format verpackt. In diesem ist nichtabdruckbare Information enthalten, so daß die Ausgabedatei nicht einfach aufgelistet oder editiert werden kann. Das genaue Format ist in Abschnitt 6 beschrieben.

Beispiel

Das folgende Programmfragment zeigt die Parameterversorgung und den Aufruf der Funktion **ts11_read_leafs()** um die bei t_conin() im Parameter fromaddr erhaltene TRANSPORTADRESSE des rufenden Partners in dessen GLOBALEN NAMEN umzuwandeln.

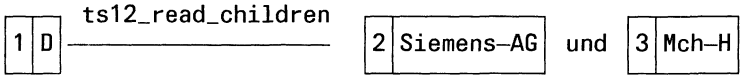
```
#include <stdio.h>
#include <cmx.h>
#include <tnsx.h>

union t_address trans;
char          npval[TS_LCOUNTRY+TS_LADMD+TS_LPRMD+TS_LOU+TS_LPN];
int           tref;

Ts_head       header = { TS_1VER01 };
Ts_leaf_name  leaf;
Ts_filter     filter[] = {
    TS_TRANS, NULL, NULL      /* ts_pr_name: TRANSPORTADRESSE */
};
Ts_P11  ts11 = {
    &header,        /* ts_head: Standardkopf */
    TS_CMX_DIR_ID,  /* ts11dir_id: TS-Directory */
    NULL,           /* ts11partial_name: vorgegebene Namenteile */
    TS_GENERAL,     /* ts11str_def: Suchmodus */
    &filter,        /* ts11filter: vorgegebene Eigenschaft */
    TS_LOCAL,       /* ts11scope: Suchbereich */
    NULL,           /* ts11filename: Ausgabedatei */
    NULL,           /* ts11time_limit: nicht unterstützt */
    npval,          /* ts11npval: Ausgabefeld für Namensteile */.1
    sizeof(npval),  /* ts11l_np_values: Länge von ts11npval */
    1,              /* ts11exp_no: 1 erwarteter GLOBALER NAMEN */
    &leaf,          /* ts11leafs: Ausgabefeld für GLOBALEN NAMEN */
    NULL            /* ts11real_no: Ausgabe gefundener Anzahl */
};
    .
    .
    .

    /*
     * Aufbauanzeige mit TRANSPORTADRESSE annehmen
     */
    if (t_conin(&tref, &lname, &trans, NULL) == T_ERROR) {
        fprintf(stderr, "CMX-Fehler %x bei t_conin\n", t_error());
        exit(1);
    }
   /*
    * TRANSPORTADRESSE in GLOBALEN NAMEN umsetzen
    */
    filter.ts_pr_length = trans.tpartaddr.t_palng;
    filter.ts_pr_value = (char *) &trans;
    if (ts11_read_leafs (&ts11) == TS_ERROR) {
        fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n",
            header.ts_retcode, header.ts_errclass, header.ts_errvalue);
        exit(1);
    }
```

```
if (ts11.ts11real_no == 0) {
    fprintf(stderr, "Keinen passenden GLOBALEN NAMEN gefunden!\n");
    exit(1);
}
 .
 .
/*
 * Der GLOBALE NAME steht nun in der Struktur leaf zur Verfügung
 */
 .
 .
 .
```

Bild 5-7 Ermittlung des GLOBALEN NAMENS mit ts_read_leaf()

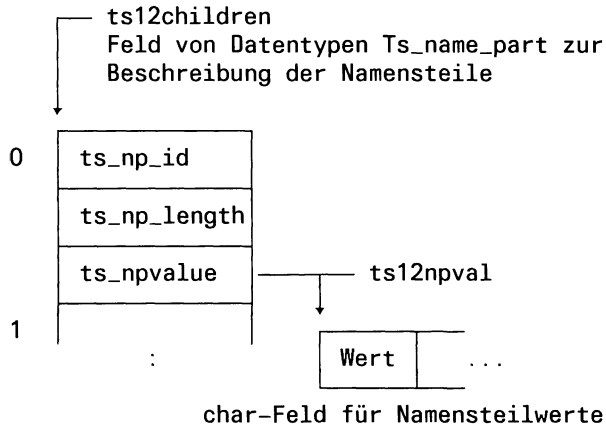
Namensbaum traversieren

Die Funktion **ts12_read_children()** bietet die Möglichkeit, den Namensbaum zu traversieren. Dazu können ausgehend vom GLOBALEN NAMEN einer NonLeafEntity alle daran angehängten Namensteile der nächsten Ebene in der Hierarchie abgefragt werden. Im Namensbaum von Bild 5-1 würde **ts12_read_children()** angewandt auf den GLOBALEN NAMEN "D" (nur Namensteil1) liefern:



Der GLOBALE NAME, von dem ausgegegangen werden soll, ist im Parameter ts12parent zu übergeben. Wie die Datenstruktur ausgefüllt werden muß, wurde bereits eingangs von Abschnitt 5 bei der "Einheitlichen Namensstruktur" beschrieben (siehe Bild 5-3 und 5-4).

Jeder zurückgelieferte Namensteil wird beschrieben durch einen Datentyp Ts_name_part. Der TNS erwartet in ts12children einen Zeiger auf ein Feld von so vielen Datentypen Ts_name_part, wie Namensteile erwartet werden. Diese Anzahl wird in ts12exp_no erwartet. Jeder Eintrag in diesem Feld nimmt die Beschreibung eines vom TNS gelieferten Namensteils auf, nämlich Identifikation, Länge und Zeiger auf den Wert des Namensteils. Für die Werte aller erwarteten Namensteile erwartet der TNS in ts12npval einen Zeiger auf ein char-Feld hinreichender Länge, die ihrerseits in ts12l_np_values anzugeben ist. In das char-Feld überträgt der TNS für jeden Namensteil dessen Wert und schreibt einen Zeiger darauf in den Eintrag des entsprechenden Namensteils unter ts12children.



Die Anzahl ts12exp_no der in ts12children bereitgestellten Feldelemente und die Länge ts12l_np_values des char-Feldes tsnpval verwendet der TNS zur Kontrolle gegen Überschreibung. Eine Formel zur Berechnung der Länge **ts12l_np_value**s des erforderlichen char-Feldes für **ts12exp_no** Namensteile ist

ts12l_np_values = ts12exp_no * TS_LPN

(vorausgesetzt, Namensteil5 (TS_PN) hat unter allen Namensteilen die größte mögliche Wertlänge).

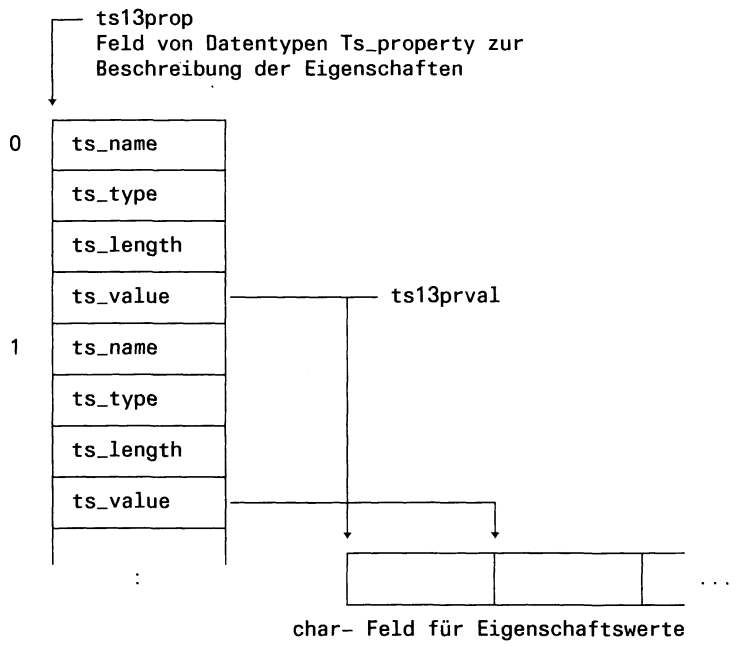
Auch bei dieser Funktion kann noch eine "Filterung" nach einer vorgegebenen Eigenschaft erfolgen, in völlig analoger Weise zu **ts11_read_leafs()** (siehe weiter oben). Das heißt, der TNS liefert dann nur solche Namensteile, die zusammen mit dem vorgegebenen GLOBALEN NAMEN der NonLeafEntity einen neuen GLOBALEN NAMEN einer LeafEntity ergeben, die diese Eigenschaft hat. Die Beschreibung des Filterkriteriums erfolgt in ts12filter wie weiter oben beschrieben (siehe Bild 5-5).

5.1.2 Eigenschaften von Anwendungen ermitteln

Die Ermittlung von Eigenschaften zu einem GLOBALEN NAMEN im TS-Directory erfolgt durch **ts13_read_properties()**.

Eigenschaften sind nur den LeafEntities im TS-Directory zugeordnet, es muß also der GLOBALE NAME einer LeafEntity angegeben werden. Dies erfolgt im Parameter ts13leaf_name. Wie die Datenstruktur ausgefüllt werden muß, wurde bereits eingangs von Abschnitt 5 bei der "Einheitlichen Namensstruktur" beschrieben (siehe Bild 5-3 und 5-4).

Jede gewünschte Eigenschaft wird durch einen Datentyp Ts_property beschrieben. Der TNS erwartet daher im Parameter ts13prop einen Zeiger auf ein Feld von so vielen Datentypen Ts_property, wie Eigenschaften erwartet werden. Jeder Eintrag in dieses Feld nimmt die Beschreibung einer vom TNS zu liefernden Eigenschaft auf. Der Eintrag ist mit der Identifikation und dem Typ der erwarteten Eigenschaft vorzubelegen (die möglichen Werte hierfür enthält Tabelle 5-3). Die Angaben zur Länge und den Zeiger zum Wert trägt dann der TNS ein. Das Feld muß mit einem Eintrag abgeschlossen werden, der als Identifikation den Wert-TS_EMPTYPROP enthält. Für die Werte aller erwarteten Eigenschaften erwartet der TNS im Parameter ts13prval einen Zeiger auf ein char-Feld hinreichender Länge, die ihrerseits im Parameter ts13l_pr_values anzugeben ist. In dieses Feld überträgt der TNS für jede Eigenschaft deren Wert und einen Zeiger darauf in den Eintrag der entsprechenden Eigenschaft in ts13prop. Ist eine gewünschte Eigenschaft nicht vorhanden, so trägt der TNS in den entsprechenden Eintrag für die Länge des Wertes NULL ein.



Die Länge ts13l_pr_values des Wertefeldes ts13prval verwendet der TNS zur Kontrolle gegen Überschreibung.

Beispiel

Das folgende Programmfragment zeigt die Parameterversorgung und den Aufruf für **ts13_read_properties()** zur Ermittlung der Eigenschaften LOKALER NAME und TRANSPORTADRESSE aus den GLOBALEN NAMEN "myleaf" und "hisleaf" für die Parameterversorgung von "name" bei t_attach() und "fromaddr" und "toaddr" bei t_conrq().

Zum Ausfüllen der erforderlichen Datenstruktur Ts_leaf_name siehe Bild 5-3 und 5-4.

```
#include <stdio.h>
#include <cmx.h>
#include <tnsx.h>

/*
 * Wertefelder für Eigenschaft TRANSPORTADRESSE, LOKALER NAME
 * und Namensteile der GLOBALEN NAMENS
 */
union t_address trans;
union t_address lname;

Ts_head      header = { TS_1VER01 } ;
Ts_leaf_name myleaf;                  /* eigener GLOBALER NAME */
Ts_leaf_name hisleaf;                 /* GLOBALER NAME Partner */
Ts_property  myname[] = {
    TS_LNAME, TS_ITEM, NULL, NULL,    /* ts_name: LOKALER NAME */
    TS_EMPTYPROP                      /* ts_name: Endekennzeichen */
};
Ts_property  paddr[] = {
    TS_TRANS, TS_ITEM, NULL, NULL,    /* ts_name: TRANSPORTADRESSE */
    TS_EMPTYPROP                      /* ts_name: Endekennzeichen */
};

Ts_P13 ts13 = {
    &header,        /* ts_head: Standardkopf */
    TS_CMX_DIR_ID,  /* ts13dir_id: TS-Directory */
    NULL,           /* ts13leaf_name: angegebener GLOBALER NAME */
    NULL,           /* ts13short_id: nicht unterstützt */
    NULL,           /* ts13prval: Wertefeld für Eigenschaften */
    sizeof (union t_address), /* ts13l_pr_values: Länge ts13prval */
    NULL,           /* ts13prop: Ausgabefeld für Eigenschaften */
    NULL,           /* ts13dist_name: nicht unterstützt */
    NULL,           /* ts13npval: nicht unterstützt */
    NULL            /* ts13l_np_values: nicht unterstützt */
};
     .
     .
     .

    /*
     * Eigener GLOBALEN NAME ist in myleaf;
     * Eigenschaft LOKALER NAME ermitteln und t_attach() aufrufen
     */
    ts13.ts13prop = myname;
```

```
ts13.ts13prval = (char *) &lname ;
ts13.ts13leaf_name = &myleaf;
if (ts13_read_properties (&ts13) == TS_ERROR) {
    fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n",
        header.ts_retcode, header.ts_errclass, header.ts_errvalue);
    exit(1);
 } else if (myname->ts_length == 0) {
    fprintf(stderr, "Kein LOKALER NAME zugeordnet!\n");
    exit(1);
}
if (t_attach(&lname, NULL) == T_ERROR) {
    fprintf(stderr, "CMX-Fehler %x bei t_attach()\n", t_error());
    exit(1);
}
/*
 * GLOBALER NAME des Partners ist in hisleaf;
 * Eigenschaft TRANSPORTADRESSE ermitteln und t_conrq() aufrufen
 */
ts13.ts13prop = paddr ;
ts13.ts13prval = (char *) &trans ;

ts13.ts13leaf_name = &hisleaf;
if (ts13_read_properties (&ts13) == TS_ERROR) {
    fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n",
        header.ts_retcode, header.ts_errclass, header.ts_errvalue);
    exit(1);
} else if (paddr->ts_length == 0) {
    fprintf(stderr, "Keine TRANSPORTADRESSE zugeordnet!\n");
    exit(1);
}
if (t_conrq(&tref, &trans, &lname, NULL) == T_ERROR) {
    fprintf(stderr, "CMX-Fehler %x bei t_conrq()\n", t_error());
    exit();
}
.
.
.
```

Bild 5-8 Ermittlung von Eigenschaften mit ts_read_properties

5.2 Informationen mit TNS verwalten

5.2.1 Anwendungen im TS-Directory einrichten

Mit folgenden Funktionen wird die interne Struktur des TS-Directory verändert, indem neue NonleafEntities und/oder LeafEntities eingerichtet oder gelöscht werden.

Funktionsaufrufe zur Informationsverwaltung	Beschreibung
ts21_create_non_leaf_entity	Einrichten einer NonLeafEntity im Namensbaum
ts22_delete_non_leaf_entity	Löschen einer NonLeafEntity im Namensbaum
ts23_create_leaf_entity	Einrichten einer LeafEntity im Namensbaum mit Eigenschaften
ts24_delete_leaf_entity	Löschen einer LeafEntity im Namensbaum

Alle Funktionen haben als einziges Argument einen spezifischen Parameterblock, dessen erster Eintrag ein Zeiger auf den Standardkopf ist (siehe Abschnitt 5). Im Standardkopf erwartet der TNS bei diesen Funktionen in ts_version den Wert TS_2VER01.

Der zweite Eintrag enthält die Identifikation des TS-Directories, auf dem die Funktion ausgeführt werden soll. TS_CMX_DIR_ID ist die Identifikation des TS-Directories für CMX.

Die Einrichtung einer NonLeafEntity erfolgt mit **ts21_create_non_leaf_entity()**. In ts21parent ist der GLOBALE NAME (Pfad im Namensbaum) anzugeben, der (ausgehend von der Wurzel des Namensbaumes) zu der neuen, noch nicht vorhandenen NonLeafEntity führen soll. Ist die NonLeafEntity bereits vorhanden, erfolgt eine Fehlermeldung. Auf diese Weise können auch an schon bestehende NonLeafEntities weitere NonLeafEntities angehängt werden. Der anzugebende GLOBALE NAME ist aber immer der Pfadname ab der Wurzel! Mit diesem Aufruf ist dann festgelegt, daß das neu im Namensbaum eingerichtete Objekt eine NonLeafEntity ist.

Die Entfernung einer NonLeafEntity aus dem Namensbaum erfolgt mit **ts22_delete_non_leaf_entity()**. In ts22parent ist der GLOBALE NAME (Pfad im Namensbaum) anzugeben, der (ausgehend von der Wurzel des Namensbaumes) zu der zu löschenden NonLeafEntity führt. Ist die NonLeafEntity nicht vorhanden bzw. gehört der GLOBALE NAME zu keiner NonLeafEntity, so erfolgt eine Fehlermeldung. An der NonLeafEntity dürfen keine weiteren Namensteile (zu NonLeafEntities oder LeafEntities) hängen.

Beispiel

Das folgende Programmfragment zeigt, wie man die NonLeafEntity im Namensbaum aus Bild 5-1

1	2	3	4	5
D	Siemens-AG	Mch-P	DF1	

erzeugt und wieder löscht.

```
#include <stdio.h>
#include <tnsx.h>

Ts_head header = { TS_2VER01 };
Ts_non_leaf_name nonleaf = {
            4,                                  /* ts_no_of_el */
    TS_COUNTRY, 1, "D",             /* ts_lf[0] */
    TS_ADMD,   10, "Siemens-AG",    /* ts_lf[1] */
    TS_PRMD,    5, "Mch-P",         /* ts_lf[2] */
    TS_OU,      3, "DF1",           /* ts_lf[3] */
};

Ts_P21 ts21 = {
    &header,    /* ts_head: Standardkopf */
    2,          /* ts21dir_id: TS-Directory */
    &nonleaf    /* ts21parent: neuer GLOBALER NAME */
};

Ts_P22 ts22 = {
    &header,    /* ts_head: Standardkopf */
    2,          /* ts22dir_id: TS-Directory */
    &nonleaf    /* ts22parent: zu löschender GLOBALER NAME */
};
```

```
/*
 * Einrichten der NonLeafEntity
 */
if (ts21_create_non_leaf_entity(&ts21) == TS_ERROR) {
    fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n",
        header.ts_retcode, header.ts_errclass, header.ts_errvalue);
    exit(1);
}

/*
 * Löschen der NonLeafEntity
 */
if (ts22_delete_non_leaf_entity(&ts22) == TS_ERROR) {
    fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n",
        header.ts_retcode, header.ts_errclass, header.ts_errvalue);
    exit(1);
}
```

Bild 5-9 Einrichten und Löschen von NonLeafEntities

Die Einrichtung einer LeafEntity erfolgt mit **ts23ecreatealeaftentity()**. Dabei können sogleich auch Eigenschaften zugewiesen werden. In ts23parent ist der GLOBALE NAME (Pfad im Namensbaum, ausgehend von der Wurzel) der NonLeafEnitity anzugeben, an der die neue, noch nicht vorhandene LeafEntity eingerichtet werden soll. Dies geschieht dann durch Anfügen des Namensteils ts23_dist_arc an den GLOBALEN NAMEN der NonLeafEntity, womit dann ein um einen Namensteil längerer GLOBALER NAME einer LeafEntity entsteht. Namensteil ts23dis_arc muß in der Hierarchie der Namenteils (siehe Einheitliche Namensstruktur in Abschnitt 5) niedriger sein, als alle im GLOBALEN NAMEN der NonLeafEntity vorhandenen Namensteile. Ist die LeafEntity bereits vorhanden, erfolgt eine Fehlermeldung.

Auf diese Weise können also an beliebige schon bestehende NonLeafEntities LeafEntities angehängt werden. Die hierarchische Reihenfolge der Namensteile ist aber zu beachten. Mit diesem Aufruf ist dann festgelegt, daß das neu im Namensbaum eingerichtete Objekt eine LeafEntity ist.

Bei der Einrichtung einer LeafEntity können ihr auch schon Eigenschaften zugewiesen werden. Beim Einrichten einer Leaf Entity muß dieser mindestens eine Eigenschaft zugeordnet werden. In ts23prop ist dabei ein Zeiger anzugeben auf ein Feld von Datentypen Ts_property, von denen jeder eine Eigenschaft beschreibt. Wie diese Datentypen auszufüllen sind, kann Bild 5-5 entnommen werden.

Die Entfernung einer LeafEntity samt ihrer Eigenschaften aus dem Namensbaum erfolgt mit **ts24_delete_leaf_entity()**. In ts24dist_name ist der GLOBALE NAME (Pfad im Namensbaum, ausgehend von der Wurzel) anzugeben, der zu der zu löschenden LeafEntity führt. Ist die LeafEntity nicht vorhanden bzw. gehört der GLOBALE NAME zu keiner LeafEntity, so erfolgt eine Fehlermeldung.

Beispiel

Das folgende Programmfragment zeigt, wie man die LeafEntity im Namensbaum aus Bild 5-1 erzeugt und wieder löscht.

1	2	3	4	5
D	Siemens-AG	Mch-P	DF1	G. Meier

Die Eigenschaften in "prop[]" für die Einrichtung der LeafEntity seien bereits versorgt (siehe Bild 5-5).

```
#include <stdio.h>
#include <cmx.h>
#include <tnsx.h>

union t_address trans;
Ts_route        rout;
Ts_head         header = { TS_2VER01 };
Ts_leaf_name    leaf = {
    5,                          /* ts_no_of_el */
    TS_COUNTRY, 1, "D",         /* ts_lf[0] */
    TS_ADMD,   10, "Siemens-AG", /* ts_lf[1] */
    TS_PRMD,    5, "Mch-P",     /* ts_lf[2] */
    TS_OU,      3, "DF1",       /* ts_lf[3] */
    TS_PN,      5, "Meier"      /* ts_lf[4] */
};
Ts_property     prop[] = {
    TS_TRANS, TS_ITEM, sizeof(trans), trans,
    TS_ROUT,  TS_ITEM, sizeof(rout),  rout,
    TS_EMPTYPROP
};
```

```
Ts_P23 ts23 = {
    &header,     /* ts_head: Standardkopf */
    2,           /* ts23dir_id: TS-Directory */
    NULL,        /* ts23parent: GLOBALER NAME NonLeafEntity */
    NULL         /* ts23dist_arc: Namensteil für LeafEntity */
    prop         /* ts23prop: Eigenschaften */
};

Ts_P24 ts24 = {
    &header,     /* ts_head: Standardkopf */
    2,           /* ts23dir_id: TS-Directory */
    NULL         /* ts24dist_name: GLOBALER NAME LeafEntity */
};
     .
     .
     .

    /*
     * Pfad von NonLeafEntity und Namensteil für LeafEntity setzen
     */
    leaf.ts_no_of_el = 4;
    ts23.ts23parent = (Ts_non_leaf_name *) &leaf;
    ts23.ts23dist_arc = (Ts_name_part *) &leaf.ts_lf[4];

    if (ts23_create_leaf_entity(&ts23) == TS_ERROR) {
        fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n",
            header.ts_retcode, header.ts_errclass, header.ts_errvalue);
        exit(1);
    }
    ...
    /*
     * Löschen der LeafEntity
     */
    leaf.ts_no_of_el = 5;
    ts24.ts24dist_name = &leaf;

    if (ts24_delete_leaf_entity(&ts24) == TS_ERROR) {
        fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n",
            header.ts_retcode, header.ts_errclass, header.ts_errvalue);
        exit(1);
    }
    .
    .
    .
```

Bild 5-10 Einrichten und Löschen von LeafEntities

5.2.2 Eigenschaften von Anwendungen einrichten, ändern, löschen

Zum Einrichten, Ändern und Löschen von NonLeafEnitities und LeafEntities im Namensbaum dienen die Funktionen:

Funktionsaufrufe zur Informationsverwaltung	Beschreibung
ts25_add_property	Einrichten einer Eigenschaft an einem GLOBALEN NAMEN
ts26_delete_property	Löschen einer Eigenschaft an einem GLOBALEN NAMEN
ts27_change_property	Ändern einer Eigenschaft an einem GLOBALEN NAMEN

Alle Funktionen haben als einziges Argument einen spezifischen Parameterblock, dessen erster Eintrag ein Zeiger auf den Standardkopf ist (siehe Abschnitt 5). Im Standardkopf erwartet der TNS bei diesen Funktionen in ts_version den Wert TS_2VER01.

Der zweite Eintrag enthält die Identifikation des TS-Directories, auf dem die Funktion ausgeführt werden soll. TS_CMX_DIR_ID ist die Identifikation des TS-Directories für CMX.

Die Funktion **ts25_add_property()** richtet bei einer LeafEntity eine neue Eigenschaft ein. In ts25dist_name ist der GLOBALE NAME der betroffenen LeafEntity anzugeben. Wie diese Datenstruktur auszufüllen ist, wurde bereits eingangs von Abschnitt 5 bei der "Einheitlichen Namensstruktur" beschrieben (siehe auch Bild 5-3 und 5-4). In ts25prty erwartet der TNS die Beschreibung der neuen Eigenschaft.

Beispiel

Das folgende Programmfragment zeigt, wie die Eigenschaft TS_LNAME der LeafEntity

1	2	3	4	5
D	Siemens-AG	Mch-P	DF1	G. Meier

zugeordnet wird.

```
#include <stdio.h>
#include <cmx.h>
#include <tnsx.h>

union t_address lname;
        Ts_head         header = { TS_2VER01 };
Ts_leaf_name    leaf = {
    5,                              /* ts_no_of_el */
    TS_COUNTRY, 1, "D",             /* ts_lf[0] */
    TS_ADMD,   10, "Siemens-AG",    /* ts_lf[1] */
    TS_PRMD,    5, "Mch-P",         /* ts_lf[2] */
    TS_OU,      3, "DF1",           /* ts_lf[3] */
    TS_PN,      8, "G._Meier"       /* ts_lf[4] */
};
Ts_property     prop = {
    TS_LNAME, TS_ITEM, NULL, &lname,
};

Ts_P25 ts25 = {
    &header,    /* ts_head: Standardkopf */
    2,          /* ts25dir_id: TS-Directory */
    &leaf,       /* ts25dist_name: GLOBALER NAME LeafEntity */
    NULL        /* ts25short_id: nicht unterstützt */
    &prop       /* ts25prty: neue Eigenschaft */
};
     .
     .
     .

    /*
     * Länge der Eigenschaft versorgen, neue Eigenschaft einrichten
     */
    prop.ts_length = lname.tmyname.t_mnlng;
    if (ts25_add_property(&ts25) == TS_ERROR) {
        fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n",
            header.ts_retcode, header.ts_errclass, header.ts_errvalue);
        exit(1);
    }
    .
    .
    .
```

Bild 5-11 Einrichten einer Eigenschaft an einer LeafEntity

Die Funktion **ts26_delete_property()** löscht an einer LeafEntity eine vorhandene Eigenschaft. In ts26dist_name ist der GLOBALE NAME der betroffenen LeafEntity anzugeben. Wie diese Datenstruktur auszufüllen ist, wurde bereits eingangs von Abschnitt 5 bei der "Einheitlichen Namensstruktur" beschrieben (siehe auch Bild 5-3 und 5-4). In ts26prty erwartet der TNS die Identifikation der neuen Eigenschaft (siehe Tabelle 5-3 für die Merkmale der Eigenschaften).

Beispiel

Das folgende Programmfragment zeigt, wie die Eigenschaft TS_LNAME der LeafEntity

1	2	3	4	5
D	Siemens-AG	Mch-P	DF1	G. Meier

gelöscht wird.

```
#include <stdio.h>
#include <tnsx.h>

Ts_head         header = { TS_2VER01 };
Ts_leaf_name    leaf = {
    5,                              /* ts_no_of_el */
    TS_COUNTRY, 1, "D",             /* ts_lf[0] */
    TS_ADMD,   10, "Siemens-AG",    /* ts_lf[1] */
    TS_PRMD,    5, "Mch-P",         /* ts_lf[2] */
    TS_OU,      3, "DF1",           /* ts_lf[3] */
    TS_PN,      5, "G. Meier"       /* ts_lf[4] */
};

Ts_P26 ts26 = {
    &header,    /* ts_head: Standardkopf */
    2,          /* ts26dir_id: TS-Directory */
    &leaf,       /* ts26dist_name: GLOBALER NAME LeafEntity */
    NULL        /* ts26short_id: nicht unterstützt */
    TS_LNAME    /* ts26prty: zu löschende Eigenschaft */
};
     .
     .
     .

    /*
     * Löschen der Eigenschaft LOKALER NAME
     */
    if (ts26_delete_property(&ts26) == TS_ERROR) {
        fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n"
            header.ts_retcode, header.ts_errclass, header.ts_errvalue);
        exit(1);
    }
    .
    .
    .
```

Bild 5-12 Löschen einer Eigenschaft einer LeafEntity

Die Funktion ts27_change_property() ändert an einer LeafEntity den Wert (und die Länge) einer vorhandenen Eigenschaft. In ts27dist_name ist der GLOBALE NAME der betroffenen LeafEntity anzugeben. Wie diese Datenstruktur auszufüllen ist, wurde bereits eingangs von Abschnitt 5 bei der "Einheitlichen Namensstruktur" beschrieben (siehe auch Bild 5-3 und 5-4). Aus dem in ts27prty übergebenen Datentypen Ts_property entnimmt der TNS die Identifikation der zu ändernden Eigenschaft, den neuen Wert und die neue Länge.

Beispiel

Das folgende Programmfragment zeigt, wie für die LeafEntity

1	2	3	4	5
D	Siemens-AG	Mch-P	DF1	G. Meier

die Eigenschaft LOKALER NAME, deren Wert in "lname" vorgegeben ist, geändert wird.

```
#include <stdio.h>
#include <cmx.h>
#include <tnsx.h>

union t_address lname;
Ts_head         header = { TS_2VER01 };
Ts_leaf_name    leaf = {
    5,                              /* ts_no_of_el */
    TS_COUNTRY, 1, "D",             /* ts_lf[0] */
    TS_ADMD,   10, "Siemens-AG",    /* ts_lf[1] */
    TS_PRMD,    5, "Mch-P",         /* ts_lf[2] */
    TS_OU,      3, "DF1",           /* ts_lf[3] */
    TS_PN,      5, "G._Meier"       /* ts_lf[4] */
};
Ts_property     prop = {
    TS_LNAME, TS_ITEM, NULL, lname,
};

Ts_P27 ts27 = {
    &header,    /* ts_head: Standardkopf */
    2,          /* ts27dir_id: TS-Directory */
    &leaf,      /* ts27dist_name: GLOBALER NAME LeafEntity */
    NULL        /* ts27short_id: nicht unterstuetzt */
    &prop       /* ts27prty: zu aendernde Eigenschaft */
};
     .
     .
     .

    /*
     * Länge der Eigenschaft in prop versorgen,
     * dann Eigenschaft ändern
     */
    prop.ts_length = lname.tmyname.t_mnlng;
    if (ts27_change_property(&ts27) == TS_ERROR) {
        fprintf(stderr, "TNS-Fehler: Typ %d, Klasse %d, Wert %d\n"0,
            header.ts_retcode, header.ts_errclass, header.ts_errvalue);
        exit(1);
    }
    .
    .
    .
```

Bild 5-13 Ändern einer Eigenschaft einer LeafEntity

6 CMX-Aufrufe, alphabetisch geordnet

Die folgenden Abschnitte enthalten die CMX-Aufrufe in alphabetischer Reihenfolge entsprechend der drei Schnittstellen ICMX(L), ICMX(TNS) und ICMX(NEA).

6.1 CMX-Aufrufe zur Kommunikation

Dieser Abschnitt enthält die Aufrufe von ICMX(L). Sie folgen auf eine zusammenfassende Einführung, die in kompakter Form die Konzeption von ICMX(L) beschreibt.

NAME

ICMX(L) - Programmschnittstelle zur Kommunikationsmethode CMX in SINIX

BESCHREIBUNG

Inhalt

Die folgenden Seiten beschreiben die Programmschnittstelle ICMX(L) zur Kommunikationsmethode CMX in SINIX. Sie enthalten

- einen Überblick über die Funktionen der Schnittstelle ICMX(L) mit Details zu den Kommunikationsphasen und Hinweisen zur korrekten Verwendung der Funktionen (Zustandsautomaten);
- die präzise Beschreibung der Funktionsaufrufe mit allen Parametern. (Es ist zu beachten, daß bei Stationskopplung die in Abschnitt 8.3 aufgeführten Einschränkungen gelten.)

Transport Service ISO IS 8072

CMX bietet in der vorliegenden Version CMX V2.1 mit ICMX(L) eine Programmschnittstelle zum verbindungsorientierten Transport Service (TS) gemäß ISO IS 8072. Der TS erlaubt es zwei Anwendungen (TS-Anwendungen), Nachrichten über eine Transportverbindung (TV) auszutauschen. Die verbindungsorientierte Kommunikation bietet Nachrichtenaustausch in zuverlässiger Weise unter Beibehaltung der Nachrichtenreihenfolge. Ferner ermöglicht der verbindungsorientierte TS über Verbindungsidentifikationen den Verzicht auf Adreßübertragung und -verarbeitung in der Datenphase. Zur korrekten Funktion der Kommunikation müssen gewisse Regeln beachtet werden, die im folgenden beschrieben werden.

ICMX(L) 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.

Gewisse Parameter, die den Transport der Nachrichten auf der TV beeinflussen, können von den TS-Anwendungen beim Verbindungsaufbau verhandelt werden.

Für die TV ist die **Transportreferenz** wesentlich. Das ist eine Zahl, die eine TV beim Auftragsaustausch zwischen CMX und TS-Anwendung eindeutig identifiziert. Intern ist jeder TV eine exklusiv eröffnete SINIX-Gerätedatei zugeordnet, die für die TS-Anwendung aber nicht sichtbar wird. Die exklusive Eröffnung vereinfacht in CMX die Aufräummaßnahmen bei vorzeitiger Beendigung der TS-Anwendung.

Fehlerbehandlung

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 als Nichteinhaltung der Kommunikationsregeln durch die TS-Anwendung erkannt werden, haben einen spezifischen Fehlercode und sind in **cmx.h** definiert. Andere Fehler resultieren aus Mißerfolgen beim Aufruf von Funktionen der Betriebssystemumgebung in CMX, sie können aus **errno.h** ermittelt werden. Die verwendeten Transportsysteme erzeugen keine Fehlermeldungen, eventuelle Fehlerfälle führen zum Verbindungsabbau mit einem entsprechenden Abbaugrund.

TS-Anwendungen, Transportverbindungen und SINIX-Prozesse

Eine TS-Anwendung ist ein System von Programmen, das den TS, also die Dienste von CMX "anwendet". Die Abbildung der TS-Anwendung auf das SINIX-Prozeßkonzept bleibt dem Implementierer überlassen. Eine TS-Anwendung kann sich in einem oder mehreren (nicht notwendig verwandten) Prozessen organisieren. Die Prozesse können prinzipiell unabhängig voneinander TVen zu fernen TS-Anwendungen unterhalten. Die Prozesse einer TS-Anwendung können ihre TVen untereinander austauschen. Die Transportreferenz einer TV ist jedoch zu jedem Zeitpunkt genau einem Prozeß zugeordnet, sie kann deshalb nicht an Kindprozesse vererbt werden. Es gibt in CMX einen eigenen lokalen Dienst REDIRECT zur Umlenkung einer TV an einen anderen Prozeß.

Ein Prozeß 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 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, wenn asynchron die Anzeige des Verbindungsabbaus eintrifft, worüber die TS-Anwendung unverzüglich informiert werden muß.

Die Funktionen von CMX 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 angenommen und der TS-Anwendung bei nächster Gelegenheit auf Anfrage zugestellt.

CMX 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 der aufrufende Prozeß suspendiert, bis ein TS-Ereignis eintrifft. Dieses weckt den Prozeß, damit er das TS-Ereignis sofort verarbeiten kann. Damit ein synchron wartender Prozeß aber nicht endlos wartet, kann er auch durch Signale wie SIGALARM geweckt werden. In diesem Falle setzt ihn CMX mit dem TS-Ereignis T_NOEVENT fort, sofern der Prozeß eine Signalbehandlung angemeldet hat. Der synchrone Mechanismus ist nützlich für TS-Anwendungen, die gleichzeitig mehrere TVen unterhalten, damit sie diese nicht pollen müssen.
Bei asynchroner Ausführung kann der Prozeß zu ihm genehmen Zeitpunkten, etwa am Ende eines Verarbeitungsschrittes, nachfragen, ob ein TS-Ereignis eingetreten ist und dieses behandeln, bevor er mit dem nächsten Verarbeitungsschritt fortfährt. Dies ist nützlich für Prozesse, die zwischen zwei TS-Ereignissen längere Pausen erwarten, in denen sie andere Verarbeitungen erledigen können oder müssen.

Die entsprechende Funktion in CMX ist t_event().

> Sie suspendiert den Prozeß bei Übergabe des Parameterwertes T_WAIT bis zum Eintreten eines TS-Ereignisses; liegt bereits ein TS-Ereignis oder Fehler vor, kehrt sie sofort mit dessen Code oder T_ERROR als Rückmeldung zurück; der suspendierte Prozeß wird beim Eintreffen eines Signals geweckt und kehrt mit T_NOEVENT oder T_ERROR zurück; 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 zehn asynchrone TS-Ereignisse sind in CMX definiert:

T_NOEVENT
kein TS-Ereignis vorhanden im asynchronen Fall, Abbruch durch Signal im synchronen Fall;

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;

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

T_DATAIN
Eintreffen gewöhnlicher Daten von der sendenden TSAnwendung;

T_XDATIN
Eintreffen von Vorrangdaten von der sendenden TS-Anwendung;

T_DATAGO
Aufheben einer durch die Flußregelung gesetzten Sendesperre für gewöhnliche Daten;

T_XDATGO
Aufheben einer durch die Flußregelung gesetzten Sendesperre für Vorrangdaten;

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.

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.

Kommunikationsphasen

Der TS in CMX unterscheidet vier Phasen: Aktivierung/Deaktivierung, Anmeldung/Abmeldung, Verbindungsauf- und -abbau, Datenaustausch. Zustandsautomaten, die zulässige Reihenfolgen für Funktionsaufrufe in diesen Phasen definieren, zeigen Bild 6-1 bis 6-6.

Aktivierung/Deaktivierung

In dieser Phase wird ein Prozeß, in dem ein TS-Anwendungsprogramm abgearbeitet wird, erzeugt oder aufgelöst.

Anmeldung/Abmeldung

In dieser Phase wird die Prozeßumgebung für die Kommunikation über CMX hergestellt, wenn die TS-Anwendung sich bei CMX zur Kommunikation anmeldet. Dabei wird ein Dienstzugriffspunkt (Transport Service Access Point TSAP) eingerichtet, an dem der TS zur Verfügung steht. Dem TSAP ist ein LOKALER NAME zugeordnet, unter dem die TS-Anwendung in diesem Endsystem erreichbar ist. Außerdem wird eine SINIX-Gerätedatei für den Austausch von Aufträgen zwischen den CMX-Bibliotheksfunktionen und dem Betriebssystem eröffnet. Bei der Abmeldung werden noch bestehende TVen und der TSAP abgebaut, die Prozeßumgebung aufgelöst und belegte Betriebsmittel für zukünftige Verwendung freigegeben.

Ein Prozeß kann gleichzeitig mehrere TSAP verwalten und an diesen mehrere Verbindungsendpunkte (Transport Connection Endpoint TCEP) unterhalten. Auch können mehrere Prozesse denselben TSAP verwenden und an diesem aktiv TVen aufbauen oder passiv auf Verbindungsanzeigen warten ohne zu interferieren. Allerdings ist jeder TCEP genau einem Prozeß 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 (den laufenden Prozeß) einer TS-Anwendung bei CMX an. Der Prozeß kann bei der Anmeldung sein künftiges Verhalten in dieser TS-Anwendung spezifizieren. Mit der ersten Anmeldung beginnt CMX für diese TS-Anwendung Verbindungsaufbauanzeigen entgegenzunehmen;

t_detach
meldet (den laufenden Prozeß) einer TS-Anwendung bei CMX ab. noch bestehende TVen des Prozesses in dieser TS-Anwendung baut CMX ab. Sofern kein weiterer Prozeß dieser TS-Anwendung angemeldet ist, ist die TS-Anwendung danach bei CMX nicht mehr bekannt.

Verbindungsauf- und -abbau

In dieser Phase bauen zwei TS-Anwendungen eine TV zueinander auf oder ab. Eine der beiden 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 ab. Während des Verbindungsaufbaus verhandeln die TS-Anwendungen gewisse Merkmale der TV für die kommende Datenphase und können Benutzerdaten austauschen.

Jede der TS-Anwendungen kann jederzeit den Abbau der TV anfordern. Dies wird von den TS-Anwendungen nicht verhandelt, sondern von CMX sofort vollzogen. Es wird der anderen TS-Anwendung (oder beiden, wenn CMX die TV abbaut) als Verbindungsabbauanzeige zugestellt, die weder beantwortet noch abgewehrt werden kann. Auch CMX kann jederzeit die TV abbauen, alle Fehlerfälle in den Transportsystemen werden auf diese Weise angezeigt. CMX garantiert nicht, daß Daten, die zum Zeitpunkt der Anforderung des Abbaus noch unterwegs sind, noch zugestellt werden.

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 die mit T_CONCF angezeigte Anwort der gerufenen TS-Anwendung; damit ist der Verbindungsaufbau vollzogen;

t_conin
übernimmt von CMX 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 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 Verbindungabbau an; sie kann jederzeit von jeder der TS-Anwendungen aufgerufen werden; mit ihr wird auch eine Verbindungsaufbauanforderung abgelehnt (statt akzeptiert) nachdem sie durch CMX angezeigt und von der TS-Anwendung übernommen wurde;

t_disin
übernimmt von CMX die mit T_DISIN angezeigte Verbindungsabbauanzeige; durch diesen Funktionsaufruf erhält die TS-Anwendung auch den Abbaugrund.

Datenaustausch, Flußregelung und Verbindungsumlenkung

Nachdem eine TV aufgebaut ist, kann über sie der Transfer von gewöhnlichen und (optional) Vorrangdaten erfolgen. Er findet nachrichtenorientiert 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 Byte beschränkt, sie werden mit Vorrang zum Strom der gewöhnlichen Daten übertragen und in eigenen Warteschlangen geführt. Pro Aufruf kann an CMX höchstens eine komplette ETSDU übergeben werden.

Die Übergabe einer TSDU an CMX muß in Portionen von maximal der Länge einer Transport Interface Data Unit (TIDU) erfolgen, wenn die TSDU länger als die TIDU ist. Die TSDU muß dann 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 garantiert nur, daß die sequentielle Zusammenfügung der TIDUs auf Empfangsseite wieder die TSDU der Sendeseite liefert. Die TIDU-Länge kann für beide TS-Anwendungen verschieden sein und hängt von der TV ab. Sie muß bei CMX für jede TV abgefragt werden. CMX garantiert nicht, daß bei der empfangenden TS-Anwendung jede außer der letzten TIDU einer TSDU maximal gefüllt zugestellt wird.

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

Die Übertragung von TIDUs (ETSDUs) unterliegt Flußregelungsmechanismen, die von CMX und den TS-Anwendungen geregelt werden können. Die Rückmeldung T_DATASTOP (T_XDATSTOP) beim Senden sagt der sendenden TS-Anwendung, daß die TIDU (ETSDU) abgesetzt, der Fluß der TIDUs (ETSDU) aber gesperrt wurde. Es können 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 TIDU (ETSDU) durch Funktionsaufrufe an CMX, die sich für die sendende TS-Anwendung wie oben auswirken.

Die Verbindungsumlenkung ist ein lokaler Dienst in CMX, der die Organisation der TS-Anwendung in Prozesse vereinfacht. Ein Prozeß, der eine fertig aufgebaute TV hält, kann diese (allerdings abhängig vom Zustand, siehe Bild 6-3 bis 6-6) an einen anderen Prozeß derselben TS-Anwendung umlenken. Der TSAP und der TCEP bleiben dabei unverändert. Der abgebende Prozeß verliert dabei die Transportreferenz dieser TV, womit sie für ihn nicht mehr verfügbar wird.

Die folgenden Funktionen realisieren Datenaustausch, Flußregelung und Verbindungsumlenkung:

t_info
liefert für eine etablierte TV die Länge der TIDU;

t_datarq
fordert die Übertragung einer (evtl. teilgefüllten) TIDU an; die Rückmeldung T_DATASTOP besagt, daß der Fluß gesperrt ist; weitere Sendeanforderungen werden mit Fehler abgewiesen, bis der Fluß wieder freigegeben wird;

t_datain
übernimmt die Daten einer TIDU von CMX, 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_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, 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_redrq
lenkt die TV an einen Prozeß derselben TS-Anwendung um, dem abgebenden Prozeß ist sie nicht mehr verfügbar;

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

t_datastop
sperrt empfangsseitig den Fluß der Normaldaten auf einer Verbindung; das Ereignis T_DATAIN wird von CMX für die Verbindung nicht mehr angezeigt;

t_datago
gibt empfangsseitig den (gesperrten) Fluß der Normaldaten und Vorrangdaten auf einer Verbindung wieder frei; das Ereignis T_DATAIN wird von CMS für diese Verbindung wieder angezeigt;

t_xdatstop
sperrt empfangsseitig den Fluß der Vorrangdaten und der Normaldaten auf einer Verbindung; CMX zeigt für diese Verbindung die Ereignisse T_DATAIN nicht mehr an;

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

Zustandsautomaten

Die Abläufe zur Nutzung der Schnittstelle ICMX(L) sind mit den folgenden Zustandsautomaten dargestellt. Das sind Diagramme, die die definierten Zustände einer TS-Anwendung und die legalen Zustandsübergänge durch die CMX-Aufrufe enthalten.

Die Abläufe können in vier Phasen aufgeteilt werden. Diese bilden eine hierarchische Struktur,

Phase A: Aktivierung/Deaktivierung
Phase B: Anmeldung/Abmeldung
Phase C: Verbindungsauf-/-abbau
Phase D: Datenaustausch

wobei die Phase A der höchsten und die Phase D der niedrigsten Hierarchiestufe entspricht.

Jede Phase wird durch einen oder mehrere Automaten dargestellt. Die doppelten Rechtecke eines Automaten stellen den Zustand dar, in dem die Automaten der nächsten Phase bzw. niedrigeren Hierarchiestufe aktiviert werden. Der Übergang aus einem Zustand, der durch ein doppeltes Rechteck dargestellt ist, in einen anderen Zustand bewirkt, daß die zugeordneten Automaten der niedrigeren Hierarchiestufe deaktiviert werden.

In Phase A muß ein Prozeß vorhanden sein, der die Schnittstelle ICMX(L) bedienen kann. In dieser Phase wird der Prozeß kreiert und zerstört.

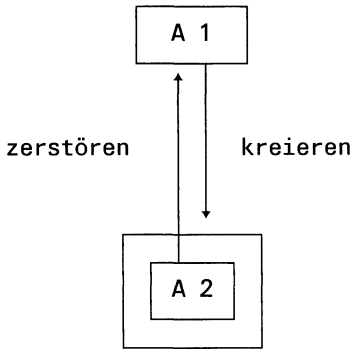


Bild 6-1 Aktivierung/Deaktivierung

In Phase B werden TS-Anwendungen an- oder abgemeldet. Die Anzahl der Automaten in dieser Phase ist gleich der Anzahl der TS-Anwendungen, die der Prozeß bedient. Im Zustand B2 ist der Prozeß in der CMXAnwendung angemeldet, evt. nicht als erster.

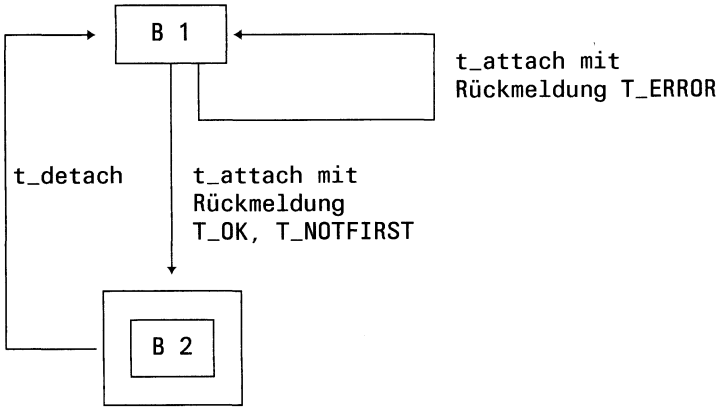


Bild 6-2 An-/Abmeldung von TS-Anwendungen

In Phase C werden TVen auf- und abgebaut und umgelenkt. Die Anzahl der Automaten ist gleich der Anzahl der TVen, die der Prozeß bedient.

Im Zustand C2 wartet der Prozeß auf die Bestätigung seines Verbindungsaufbaues. Im Zustand C3 nimmt der Prozeß eine Verbindungsaufbauanforderung entgegen. Im Zustand C4 ist die Verbindung vollständig aufgebaut.

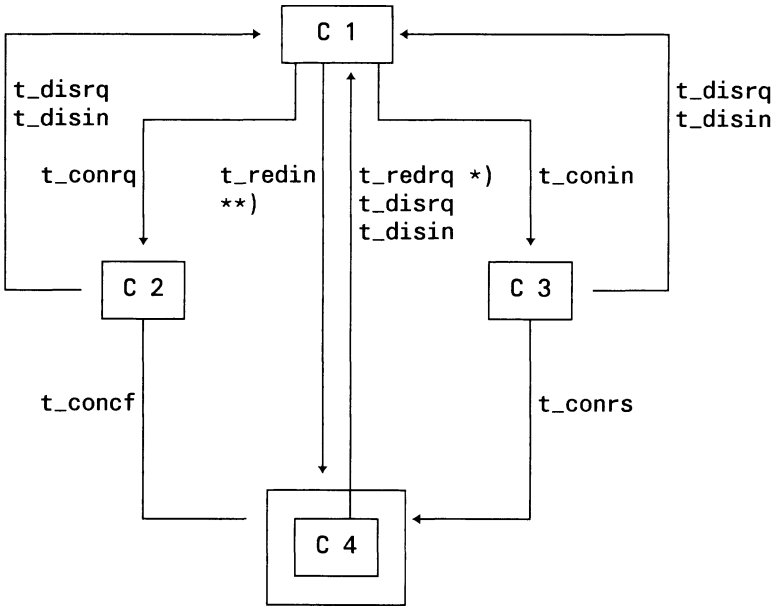


Bild 6-3 Auf- und Abbau und Umlenkung von TVen

*) t_redrq ist nur dann erlaubt, wenn der entsprechende Datensendeautomat in D1 und die Datenempfangsautomaten in D1 oder D4 sind.

**) t_redin bewirkt, daß die Datenautomaten in die Zustände übergehen, die den Zuständen der Datenautomaten des Prozesses entsprechen, der das t_redrq initiiert hat.

Phase D ist die Datenphase. Sie wird durch 2 parallele Automaten (Datensendeautomat, Datenempfangsautomat) dargestellt. Es gibt also n Automaten, wobei n = 2 * Anzahl der TVen ist.

Im Zustand D2 liegt die Sperre des sendeseitigen Flusses von Normaldaten vor, im Zustand D3 die des Flusses von Vorrangdaten.

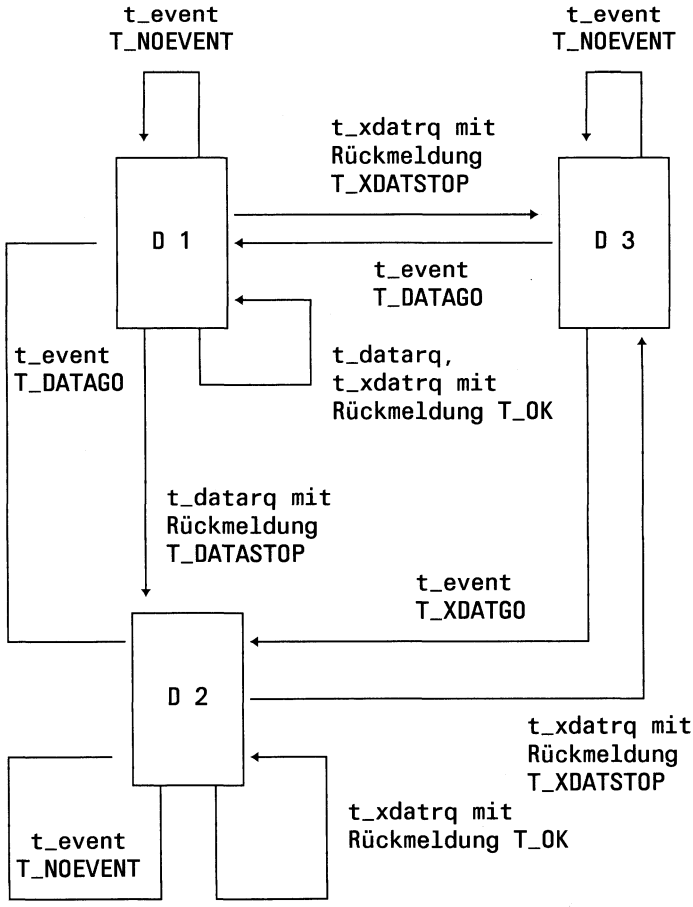


Bild 6-4 Datensendeautomat für Normal- und Vorrangdaten

Der Datenempfangsautomat wird der Übersichtlichkeit halber in zwei getrennten Automaten (für Normaldaten bzw. für Vorrangdaten) dargestellt. Es ist zu beachten, daß t_event(), t_xdatstop() und t_datago() auf beide Automaten wirken. Zustand D4 wird eingenommen, wenn der Prozeß den empfangsseitigen Fluß der Normal- oder Vorrangsdaten sperrt.

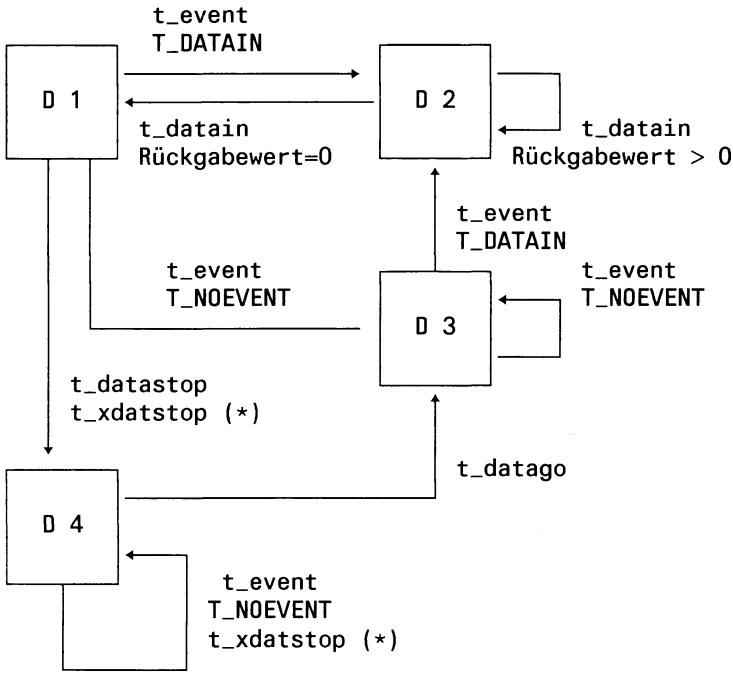


Bild 6-5 Datenempfangsautomat für Normaldaten

(*) t_xdatstop ist nur erlaubt, wenn der Datenempfangsautomat für Vorrangdaten in D1 ist.

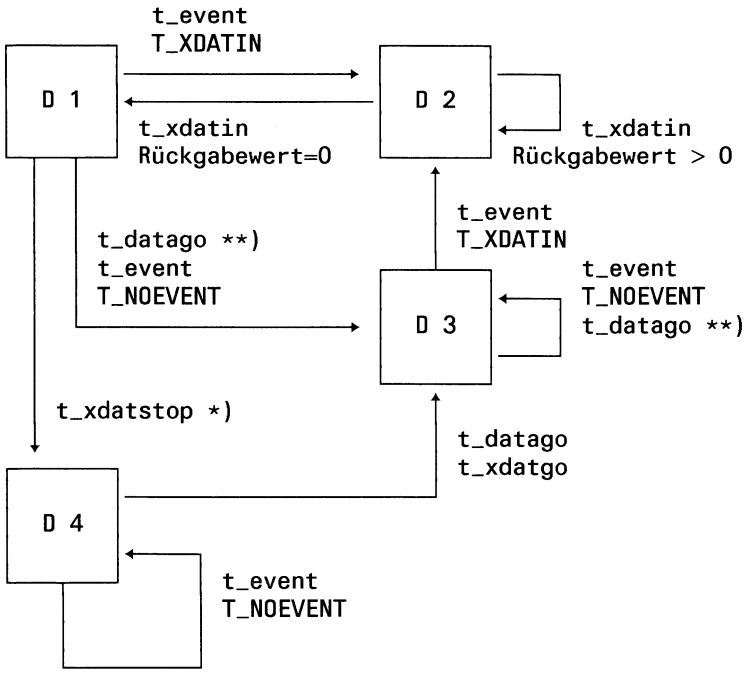


Bild 6-6 Datenempfangsautomat für Vorrangdaten

*) t_xdatstop ist nur erlaubt, wenn der Datenempfangsautomat für Normaldaten in D1 oder D4 ist.

**) t_datago ist nur erlaubt, wenn der Datenempfangsautomat für Normaldaten in D4 ist.

Programmierhinweise

Das Hauptziel von ICMX(L) 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(L) 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 Adressierung sollte die TS-Anwendung ausschließlich mit Hilfe des TNS behandeln, sie sollte keine physischen Transportadressen in den Programmen aufbauen.

4) CMX-Funktionen sollten nicht in Signalbehandlungsroutinen aufgerufen werden, die Signalbehandlung ist nicht dazu geschaffen, asynchrone Verarbeitung 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

t_attach();
t_conrq();
for (;;) {
    switch(t_event())
    case T_CONCF:
        t_concf();
          :
          :
        t_datarq();
          :
          :
    case T_DATAIN:
        t_datain();
          :
          :
    case T_DISIN:
        t_disin();
        t_detach();
          :
    case T_NOEVENT:
        continue;

    case T_ERROR:
        t_detach();
        exit();
        default:
           :
        }
```

```
gerufene TS-Anwendung

t_attach();

for (;;) {
    switch (t_event()) {
    case T_CONIN:
        t_conin();
        t_conrs();
          :
    case T_DATAIN:
         t_datain();
          :
         t_datarq();
          :
         t_disrq();
          :
    case T_DISIN:
         t_disin();
         t_detach();
          :
    case T_NOEVENT:
         continue;

    case T_ERROR:
        t_detach();
        exit();
        default:
           :
        }
```

Konventionen

Bei Verwendung von ICMX(L) sind folgende Konventionen einzuhalten:

1) alle Identifier, die mit '_' beginnen, sind reserviert für die Systemsoftware;

2) alle Identifier, die mit "t_", "ts", "Ts" oder "cmx" beginnen sind für CMX reserviert;

3) alle Präprozessordefinitionen, die mit "T_" oder "TS" beginnen, sind für CMX reserviert;

4) das Signal SIGTERM wird von den CMX-Komponenten im Kernel verschickt und in der CMX-Bibliothek eingefangen (Signalbehandlungsroutine); es sollte nicht für andere Zwecke verwendet werden.

CMX-Funktionsaufrufe

Die folgenden Seiten beschreiben die CMX-Aufrufe im Detail. Parameter in denen CMX einen vom Aufrufer bereitgestellten Wert erwartet, sind mit "->" gekennzeichnet; Parameter, in denen CMX nach dem Aufruf einen Wert liefert, sind mit "< " gekennzeichnet; Parameter, in denen der Aufrufer einen Wert bereitstellen muß, der dann von CMX modifiziert wird, tragen das Kennzeichen "<>".

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

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

NAME

t_attach - Anmelden eines Prozesses bei CMX, attach process

SYNOPSIS

```
#include <cmx.h>

int t_attach(name, opt)
char *name;
union {struct t_opta1 opta1;} *opt;
```

BESCHREIBUNG

t_attach() meldet den laufenden Prozeß bei CMX an. Dabei ist der LOKALE NAME der TS-Anwendung als Parameter anzugeben. Den LOKALEN NAMEN liefert der TNS als eine der Eigenschaften zum GLOBALEN NAMEN der TS-Anwendung. Ist dieser Prozeß der erste, der sich mit diesem LOKALEN NAMEN anmeldet, entsteht eine TS-Anwendung. Weitere Prozesse können sich unter demselben LOKALEN NAMEN anmelden und gehören dann zur selben TS-Anwendung.
t_attach() kann einen Prozeß auch mehrmals mit verschiedenen LOKALEN NAMEN anmelden. Dann gehört der Prozeß zu verschiedenen TS-Anwendungen. **t_attach()** legt anwendungsspezifische Parameter für den Prozeß fest:

- den LOKALEN NAMEN,
- welche Arten des Verbindungsaufbaus dem Prozeß in dieser TS-Anwendung möglich sind,
- die Anzahl der möglichen Verbindungen des Prozesses in dieser TS-Anwendung.

Nach der ersten erfolgreichen Anmeldung für eine TS-Anwendung beginnt CMX für diese TS-Anwendung Verbindungswünsche entgegenzunehmen.

Parameter:

-> name
Zeiger zu einem Datenbereich mit dem LOKALEN NAMEN der TS-Anwendung. Den LOKALEN NAMEN liefert der TNS als Eigenschaft zum GLOBALEN NAMEN der TS-Anwendung.

-> opt
Angabe NULL oder Zeiger zu einer Variante mit Benutzeroptionen. CMX setzt bei Angabe NULL die Standardwerte. Folgende Benutzeroptionen sind definiert (siehe cmx.h):

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

t_optnr
Optionsnummer. Anzugeben ist T_OPTA1

t_apmode
legt fest, welche Arten von Verbindungsaufbauten dem Prozeß in dieser TS-Anwendung möglich sind. Zulässige Werte sind:

T_ACTIVE,	falls der Prozeß aktiv Verbindungen aufbauen soll,
T_PASSIVE,	falls der Prozeß passiv auf Verbindungswünsche warten soll,
T_REDIRECT,	falls der Prozeß umgelenkte Verbindungen annehmen soll.

Diese Werte können durch bitweises 'oder' kombiniert werden.
Standardwert: T_ACTIVE | T_PASSIVE | T_REDIRECT

t_conlim
Maximalzahl der Verbindungen, die dieser Prozeß gleichzeitig haben darf.
Die Angabe von tnconlim = -1 besagt, daß der Prozeß das installationsspezifische Maximum an Verbindungen gleichzeitig haben darf.
Standardwert: 1

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich. Der Prozeß hat sich als erster mit diesem Namen angemeldet.

T_NOTFIRST
Aufruf war erfolgreich. Der Prozeß hat sich als weiterer Prozeß der TS-Anwendung angemeldet.

T_ERROR
Fehler. Fehlercode mit t_error() abfragen.

Hinweis

1. Beim ersten t_attach() wird im laufenden Prozeß eine Dateikennzahl [CES] belegt und eine Behandlungsfunktion für SIGTERM [CES] angemeldet. Bei Rechnerkopplung bleibt während der Lebensdauer des Prozesses die Dateikennzahl belegt, die Signalbehandlung angemeldet. Bei Stationskopplung wird beim letzten t_detach() die Dateikennzahl wieder freigegeben und die Behandlung von SIGTERM auf SIG_IGN gesetzt.

2. Verbindungsanzeigen (ankommende Rufe) werden zunächst demjenigen Prozeß zugestellt, der sich als erster mit T_PASSIVE angemeldet hat. Ist die angegebene Verbindungsgrenze t_conlim erreicht, werden die Verbindungsanzeigen anderen Prozessen zugestellt, die sich mit demselben Namen und T_PASSIV angemeldet haben. In welcher Reihenfolge die einzelnen Prozesse die Verbindungsanzeigen zugestellt bekommen, ist nicht definiert.

3. Die Eigenschaften TS_NEABX und TS_GTYPE werden von CMX bei Stationskopplung in t_attach() implizit durch Aufrufe von ICMX(TNS) ermittelt. Falls TS_NEABX eine Länge ungleich Null und den Wert TS_TRUE hat, aktiviert CMX für die TS-Anwendung die Migrationsunterstützung. Falls die Länge von TS_GTYPE ungleich Null ist, so wird der Wert als für die TS-Anwendung zu verwendendes Geräteprotokoll angesehen, anderenfalls verwendet CMX kein Geräteprotokoll.

NAME

t_concf - Verbindung herstellen, connect confirmation

SYNOPSIS

```
#include <cmx.h>

int t_concf(tref,opt)
int *tref;
union {struct t_optc1 optc1;} *opt;
```

BESCHREIBUNG

t_concf() nimmt ein zuvor mit **t_event()** gemeldetes Ereignis T_CONCF entgegen. T_CONCF zeigt an, daß die gerufene TS-Anwendung einen Verbindungswunsch **(t_conrq())** des laufenden Prozesses positiv beantwortet hat. **t_concf()** liefert die Benutzerdaten, die die gerufene TS-Anwendung mitgeschickt hat, falls das verwendete Transportsystem diese Option bietet. Wenn der Aufruf erfolgreich war, ist die Verbindung fertig aufgebaut.

Parameter:

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

<> opt
Angabe NULL oder Zeiger zu einer Variante mit Systemoptionen. Damit werden die Benutzerdaten entgegengenommen, die die gerufene TS-Anwendung bei der Antwort auf den Verbindungswunsch mitgegeben hat. Bei Angabe NULL wirft CMX die Benutzerdaten weg. Hat die gerufene TS-Anwendung nichts angegeben, setzt CMX die 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 auf einen Datenbereich, in den CMX die empfangenen Benutzerdaten der gerufenen TS-Anwendung einträgt. Der Bereich muß so groß sein, daß die empfangenen Daten ganz hineinpassen. Die zulässige Länge hängt vom verwendeten Transportsystem ab. In cmx.h steht als für alle Transportsysteme geeignete Maximalgröße T_MSG_SIZE.

t_udatal
Vor dem Aufruf muß hier die Länge des Datenbereiches t_udatap stehen. CMX liefert in diesem Feld die Anzahl der empfangenen Byte zurück.
Standardwert: 0.

t_xdata
CMX liefert hier die Antwort der gerufenen TS-Anwendung, wenn beim Verbindungsaufbau der Austausch von Vorrangdaten vorgeschlagen wurde. T_YES bedeutet, daß die gerufene TS-Anwendung dem Vorschlag zustimmt, T_NO bedeutet, daß sie den Vorschlag ablehnt. Die Antwort ist verbindlich.
Standardwert: T_NO.

t_timeout
Der Inhalt dieses Feldes ist stets T_NO.

RÜCKMELDUNG

T_OK
Aufruf erfolgreich.

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

Hinweis

Sobald eine Verbindung etabliert ist, liegt die Initiative bei der TS-Anwendung (nicht bei CMX): sie kann sowohl Normaldaten als auch Vorrangdaten (falls vereinbart) senden oder durch **t_event()** anzeigen, daß sie bereit ist Normaldaten bzw. Vorrangdaten (falls vereinbart) zu empfangen.

NAME

t_conin - Verbindungswunsch entgegennehmen, connect indication

SYNOPSIS

```
#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;
```

BESCHREIBUNG

t_conin() nimmt ein zuvor mit **t_event()** gemeldetes Ereignis T_CONIN entgegen. T_CONIN zeigt an, daß eine rufende TS-Anwendung eine Verbindung zum laufenden Prozeß aufbauen will. Der Aufruf liefert die TRANSPORTADRESSE der rufenden TS-Anwendung und den LOKALEN NAMEN der lokalen TS-Anwendung sowie die Benutzerdaten, die die rufende TS-Anwendung mitgegeben hat.

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

Parameter:

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

<- toaddr
Zeiger zu einer Variante, in der CMX den bei t_attach() angegebenen LOKALEN NAMEN der lokalen TS-Anwendung zurückliefert. Diese Information ist wichtig, wenn ein Prozeß mehrere TS-Anwendungen steuert. Sie gibt an, welche TS-Anwendung die Verbindung erhalten soll. Der LOKALE NAME kann mit TNS in den GLOBALEN NAMEN der lokalen TS-Anwendung übersetzt werden.

<- fromaddr
Zeiger zu einer Variante, in der CMX die TRANSPORTADRESSE der rufenden TS-Anwendung zurückliefert. Die TRANSPORTADRESSE kann mit TNS in den GLOBALEN NAMEN der rufenden TS-Anwendung übersetzt werden.

<> opt
Zeiger zu einer Variante mit Systemoptionen oder Angabe NULL. Damit können die Benutzerdaten abgefragt werden, die die rufende TS-Anwendung beim Verbindungsaufbau angegeben hat. Bei Angabe NULL wirft CMX die Benutzerdaten weg. Hat die rufende TS-Anwendung nichts angegeben, liefert CMX die 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 die empfangenen Benutzerdaten der rufenden TS-Anwendung einträgt. Der Bereich muß so groß sein, daß die empfangenen Daten ganz hineinpassen. Die maximal empfangbare Länge der Benutzerdaten hängt ab vom Transportsystem. Die für alle Transportsysteme gültige Obergrenze ist in cmx.h als T_MSG_SIZE definiert.

t_udatal
Vor dem Aufruf muß hier die Länge des Datenbereiches t_udatap stehen. CMX liefert in diesem Feld die Anzahl der empfangenen Byte zurück.
Standardwert: 0.

t_xdata
CMX liefert hier den Vorschlag der rufenden TS-Anwendung nach Vorrangdaten zurück. T_YES bedeutet Vorrangdaten erwünscht, T_NO bedeutet Vorrangdaten ausgeschlossen. Schlägt die rufende TS-Anwendung Vorrangdaten vor, so ist die Antwort des laufenden Prozesses beim folgenden **t_conrs()** endgültig. Wünscht die rufende TS-Anwendung keine Vorrangdaten, können vom laufenden Prozeß beim folgenden **t_conrs()** auch keine verlangt werden. Gegebenenfalls muß dann der Verbindungswunsch mit **t_disrq()** abgelehnt werden.
Standardwert: T_NO.

t_timeout
Der Inhalt dieses Feldes ist stets T_NO.

RÜCKMELDUNG

T_OK
Aufruf erfolgreich.

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

NAME

t_conrq - Verbindung anfordern, connection request

SYNOPSIS

```
#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;} *opt;
```

BESCHREIBUNG

t_conrq() fordert den Aufbau einer Transportverbindung von der lokalen TS-Anwendung zu einer gerufenen TS-Anwendung an. Die gerufene TS-Anwendung erhält diese Anforderung als Verbindungsaufbauanzeige. Mit der Anforderung können der gerufenen TS-Anwendung Benutzerdaten mitgeschickt werden, falls das verwendete Transportsystem diese Option bietet.

Die Antwort der gerufenen TS-Anwendung wird als Ereignis T_CONCF oder T_DISIN gemeldet.

Parameter:

<- tref
Zeiger zu einem Feld, in dem CMX die verbindungsspezifische Transportreferenz zurückliefert; sie kennzeichnet die Verbindung eindeutig. Sie muß bei allen Aufrufen, die diese Verbindung betreffen, angegeben werden.

-> toaddr
Zeiger zu einer Variante, in dem die TRANSPORTADRESSE der gerufenen TS-Anwendung anzugeben ist. Die TRANSPORTADRESSE liefert der TNS als Eigenschaft zum GLOBALEN NAMEN der gerufenen TS-Anwendung.

-> fromaddr
Zeiger zu einer Variante, in dem der LOKALE NAME der lokalen TS-Anwendung anzugeben ist. Hier muß derselbe LOKALE NAME angegeben werden wie beim **t_attach()** für diese TS-Anwendung.

-> opt
Zeiger zu einer Variante mit Systemoptionen oder die Angabe NULL. Damit werden die Benutzerdaten angegeben, die die gerufene TS-Anwendung mit der Anzeige des Verbindungswunsches erhalten soll.
Bei Angabe NULL nimmt CMX die 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 Bereich mit Benutzerdaten, die die gerufene TS-Anwendung erhalten soll.
Standardwert: undefiniert

t_udatal
Länge der aus dem Bereich t_udatap zu übertragenden Benutzerdaten.
Standardwert: 0, Maximalwert: T_MSG_SIZE

t_xdata
Vorschlag an die gerufene TS-Anwendung, ob Vorrangdaten ausgetauscht werden sollen. Zulässige Werte sind T_YES, falls Vorrangdaten vorgeschlagen werden und T_NO, falls Vorrangdaten ausgeschlossen werden.
Standardwert: T_NO.

t_timeout
Angabe der Inaktiv-Zeit der Verbindung. T_NO bedeutet keine Überwachung. Ein Wert n > 0 bedeutet, daß die Verbindung n Sekunden inaktiv sein darf. Danach baut CMX sie ab.
Standardwert: T_NO.

Hinweis
Bei Stationskopplung gibt es Einschränkungen, siehe Abschnitt 8.3.

RÜCKMELDUNG

T_OK
Aufruf erfolgreich.

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

Hinweis

Beim Verbindungsaufbau über WAN- oder LAN-Kopplung wertet CMX intern die Eigenschaft TS_ROUT der TS-Anwendung aus. Ist die Eigenschaft nicht vorhanden, so wird ein Standardwert verwendet. Dieser bewirkt, daß der Verbindungsaufbau über das erste geeignete Transportsystem, zu dem der LOKALE NAME in fromaddr paßt, abgewickelt wird.

NAME

t_conrs - Verbindungswunsch beantworten, connection response

SYNOPSIS

```
#include <cmx.h>

int t_conrs(tref,opt)
int *tref;
union {struct t_optc1 optc1;} *opt;
```

BESCHREIBUNG

t_conrs() ist die positive Antwort auf einen zuvor mit **t_conin()** entgegengenommenen Verbindungsaufbauwunsch einer rufenden TS-Anwendung. Damit wird der Verbindungswunsch akzeptiert. Die rufende TS-Anwendung erhält diese Antwort als Ereignis T_CONCF. Mit der Antwort können der rufenden TS-Anwendung Benutzerdaten mitgeschickt werden, falls das verwendete Transportsystem diese Option bietet. Wenn der Aufruf erfolgreich war, ist die Verbindung für den laufenden Prozeß fertig aufgebaut.

Parameter:

-> tref
Zeiger zu einem Feld mit der beim entsprechenden **t_conin()** verwendeten Transportreferenz.

-> opt
Zeiger zu einer Variante mit Systemoptionen oder die Angabe NULL. Damit werden die Benutzerdaten angegeben, die die rufende TS-Anwendung mit der Antwort auf den Verbindungswunsch erhalten soll. Bei Angabe NULL nimmt CMX die 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 Bereich mit Benutzerdaten, die die rufende TS-Anwendung erhalten soll.
Standardwert: undefiniert

t_udatal
Länge der aus dem Bereich t_udatap zu übertragenden Benutzerdaten in Byte.
Standardwert: 0, Maximalwert: T_MSG_SIZE.

t_xdata
Antwort auf den Vorschlag der rufenden TS-Anwendung nach Vorrangdaten. T_YES bedeutet, daß der Vorschlag der rufenden TS-Anwendung nach Vorrangdaten akzeptiert wird. T_NO bedeutet, daß Vorrangdaten abgelehnt werden. Die Antwort ist verbindlich. Hat die rufende TS-Anwendung die Verwendung von Vorrangdaten von vornherein ausgeschlossen, muß mit T_NO geantwortet werden.
Standardwert: T_NO.

t_timeout
Angabe der Inaktiv-Zeit der Verbindung. T_NO bedeutet keine Überwachung. Ein Wert n > 0, daß die Verbindung n Sekunden inaktiv sein darf. Danach baut CMX sie ab.
Standardwert: T_NO.

Hinweis
Bei Stationskopplung gibt es Einschränkungen, siehe Abschnitt 8.3.

RÜCKMELDUNG

T_OK
Aufruf erfolgreich.

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

Hinweis

Sobald eine Verbindung etabliert ist, liegt die Initiative bei der TS-Anwendung (nicht bei CMX): sie kann sowohl Normaldaten als auch Vorrangdaten (falls vereinbart) senden oder durch **t_event()** anzeigen, daß sie bereit ist Normaldaten bzw. Vorrangdaten (falls vereinbart) zu empfangen.

NAME

t_datago - Datenfluß freigeben, datago

SYNOPSIS

```
#include <cmx.h>

int t_datago(tref)
int *tref;
```

BESCHREIBUNG

t_datago() gibt den gesperrten Datenfluß auf der angegebenen Verbindung frei. Der laufende Prozeß teilt CMX damit mit, daß er wieder bereit ist, Daten zu empfangen. Dieser Aufruf gibt auch den Vorrangdatenfluß (sofern vereinbart) wieder frei, falls er (auch) gesperrt war. Der Aufruf bewirkt im einzelnen:

- der laufende Prozeß erhält wieder die Ereignisse T_DATAIN und T_XDATIN für die angegebene Verbindung zugestellt, falls sie anstehen,
- die sendende TS-Anwendung erhält das Ereignis T_DATAGO zugestellt, sie darf wieder Daten senden.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich.

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

NAME

t_datain - Daten empfangen, data indication

SYNOPSIS

```
#include <cmx.h>

int t_datain(tref, datap, datal, chain)
int *tref;
char *datap;
int *datal;
int *chain;
```

BESCHREIBUNG

t_datain() nimmt ein zuvor mit **t_event()** gemeldetes Ereignis T_DATAIN entgegen. Der laufende Prozeß empfängt damit auf der angegebenen Verbindung eine Dateneinheit von der sendenden TS-Anwendung. Der in chain zurückgelieferte Indikator zeigt an, ob noch weitere Dateneinheiten zur Nachricht gehören (T_MORE) oder nicht (T_END). Jede weitere Nachricht zeigt CMX mit einem eigenen T_DATAIN an. Wenn die Dateneinheit in den mit datap und datal bereitgestellten Bereich hineinpaßt, kehrt der Aufruf mit T_OK zurück. Wenn nicht, liefert der Aufruf einen Wert n > 0. Dies ist die Anzahl der Byte, die aus der Dateneinheit noch nicht gelesen wurden (Restlänge). In diesem Fall muß **t_datain()** solange aufgerufen werden, bis die ganze Dateneinheit gelesen ist. Erst dann sind wieder andere CMX-Aufrufe, z.B. **t_event()** möglich. Die maximale Länge der Dateneinheit hängt ab vom verwendeten Transportsystem. Sie muß für eine etablierte Verbindung mit **t_info()** abgefragt werden.

Parameter:

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

<- datap
Zeiger auf einen Bereich, in den CMX die empfangenen Daten einträgt.

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

<- chain
Zeiger zu einem Indikator, in dem CMX anzeigt, ob noch weitere Dateneinheiten zur Nachricht gehören. Bei T_MORE folgt noch eine weitere zur Nachricht gehörende Nachricht, sie wird mit einem eigenen T_DATAIN angezeigt. Bei T_END ist die vorliegende Dateneinheit die letzte der Nachricht.

RÜCKMELDUNG

T_OK
Aufruf erfolgreich, Dateneinheit vollständig gelesen.

n > 0
Es sind noch n Byte in der Dateneinheit vorhanden. Sie müssen zuerst mit weiteren Aufrufen **t_datain()** abgeholt werden, bevor mit **t_event()** das nächste Ereignis geprüft werden kann.

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

NAME

t_datarq - Daten senden, data request

SYNOPSIS

```
#include <cmx.h>

int t_datarq(tref, datap, datal, chain)
int *tref;
char *datap;
int *datal;
int *chain;
```

BESCHREIBUNG

t_datarq() sendet auf der angegebenen Verbindung eine Dateneinheit an die empfangende TS-Anwendung. Die maximale Länge der Dateneinheit hängt ab vom verwendeten Transportsystem. Sie muß für eine etablierte Verbindung mit **t_info()** abgefragt werden. Wenn die zu sendende Nachricht länger ist als eine Dateneinheit, muß sie mit mehreren Aufrufen **t_datarq()** hintereinander übermittelt werden. Über den Indikator **chain** teilt der Prozeß bei jeder Dateneinheit mit, ob noch weitere Dateneinheiten zur Nachricht gehören. Bei Rückmeldung T_DATASTOP ist die Dateneinheit übernommen, der Datenfluß aber für diese Verbindung gesperrt worden. Dies kann auf Initiative der empfangenden TS-Anwendung geschehen (mit **t_datastop()**) oder auf Initiative von CMX, wenn der lokale Zwischenspeicher voll ist. Es muß dann mit **t_event()** erst das Ereignis T_DATAGO für die Verbindung abgewartet werden, bevor die nächste Dateneinheit gesendet werden kann.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

-> datap
Zeiger auf einen Bereich, in dem die zu sendenden Daten stehen.

-> datal
Zeiger zu einem Feld mit der Anzahl der zu sendenden Byte aus dem Bereich **datap** (mindestens 1). Die maximale Länge ist eine Dateneinheit.

-> chain
Zeiger zu einem Indikator, der CMX anzeigt, ob noch weitere Dateneinheiten zur Nachricht gehören. T_MORE bedeutet, es folgt noch eine weitere Dateneinheit zur Nachricht. T_END bedeutet, die vorliegende Dateneinheit ist die letzte der Nachricht.

RÜCKMELDUNG

T_OK
Aufruf erfolgreich, weitere Dateneinheiten können sofort gesendet werden.

T_DATASTOP
Aufruf erfolgreich, weitere Dateneinheiten dürfen erst gesendet werden, wenn für diese Verbindung das Ereignis T_DATAGO eingetroffen ist.

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

NAME

t_datastop - Datenfluß sperren, data stop

SYNOPSIS

```
#include <cmx.h>

int t_datastop(tref)
int *tref;
```

BESCHREIBUNG

t_datastop() sperrt den Datenfluß auf der angegebenen Verbindung. Der laufende Prozeß teilt CMX dadurch mit, daß er nicht bereit ist, für diese Verbindung Daten zu empfangen. Ein bereits angezeigtes Ereignis T_DATAIN muß aber erst bearbeitet werden. Dieser Aufruf bewirkt, daß dieser Prozeß das Ereignis T_DATAIN für die angegebene Verbindung nicht mehr zugestellt bekommt. Er kann aber währenddessen andere CMX-Funktionen aufrufen, z.B. eine weitere Verbindung aufbauen. Die sendende TS-Anwendung erhält bei **t_datarq()** das Ergebnis T_DATASTOP. Sie darf keine Daten mehr senden. Freigegeben wird der Datenfluß mit **t_datago()**. Vorrangdaten sind von t_datastop () nicht betroffen.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich.

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

NAME

t_detach - Abmelden eines Prozesses bei CMX, detach process

SYNOPSIS

```
#include <cmx.h>

int t_detach(name)
char *name;
```

BESCHREIBUNG

t_detach() meldet den laufenden Prozeß für die angegebene TS-Anwendung bei CMX ab. Falls noch Verbindungen existieren, werden sie implizit abgebaut. Meldet sich der letzte Prozeß einer TS-Anwendung ab, so ist die TS-Anwendung nicht mehr bekannt. Verbindungswünsche für diese TS-Anwendung werden dann nicht mehr angenommen.

Parameter:

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

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich.

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

NAME

t_disin - Verbindungsabbau entgegennehmen, disconnection indication

SYNOPSIS

```
#include <cmx.h>

int t_disin(tref, reason, opt)
int *tref;
int *reason;
union {struct t_optc2 optc2;} *opt;
```

BESCHREIBUNG

t_disin() nimmt ein zuvor bei **t_event()** gemeldetes Ereignis T_DISIN entgegen. T_DISIN zeigt an, daß entweder die ferne TS-Anwendung oder CMX die Verbindung abgebaut haben. Der Aufruf liefert den Grund für den Abbau der Transportverbindung. Er liefert ferner die Benutzerdaten, die die ferne TS-Anwendung beim Verbindungsabbau mitgeschickt hat, falls das verwendete Transportsystem diese Option bietet.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

<- reason
Zeiger zu einem Feld, in das CMX den Grund des Verbindungabbaus einträgt. Mögliche Werte sind weiter unten aufgeführt.

<> opt
Zeiger zu einer Variante mit Systemoptionen oder Angabe NULL. Damit können die Benutzerdaten abgefragt werden, die die ferne TS-Anwendung beim Verbindungsabbau angegeben hat. Bei Angabe NULL wirft CMX die Benutzerdaten weg. Hat die ferne TS-Anwendung nichts angegeben, liefert CMX die 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 Datenbereich, in den CMX die empfangenen Benutzerdaten der fernen TS-Anwendung einträgt. Der Bereich muß so groß sein, daß die empfangenen Daten ganz hineinpassen. Die maximal zulässige Länge der Benutzerdaten hängt vom verwendeten Transportsystem ab.

t_udatal
Vor dem Aufruf muß hier die Länge des Datenbereiches t_udatap stehen. CMX liefert in diesem Feld die Anzahl der empfangenen Byte zurück.
Standardwert: 0.

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich.

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

ABBAUGRÜNDE

Die in **reason**gelieferten Verbindungsabbaugründe haben die folgende Bedeutung (die hier angegebenen symbolischen Werte sind in cmx.h definiert, im Zweifelsfall gilt der in cmx.h definierte numerische Wert). Die Abkürzung CCP steht hier für "Communication Control Program" und gemeint ist damit das Transportsystem.

T_USER	0	Abbau durch die ferne Anwendung
T_RTIMEOUT	1	Abbau lokal durch CMX wegen Inaktivität der Verbindung gemäß Parameter t_timeout
T_RADMIN	2	Abbau lokal durch CMX wegen Außerbetriebnahme des CCP durch die Administration
T_RCCPEND	3	Abbau lokal durch CMX wegen CCP-Ausfall
T_RCCP	256	bei allen reasons ⩾ T_RCCP ist Verbindungsabbau vom CCP veranlaßt; alle diese reasons haben die Form T_RCCP + ITRANS-reason
T_RUNKNOWN	256	Abbau vom Partner-CCP, Grund nicht angegeben
T_RSAPCONGEST	257	Abbau vom Partner-CCP wegen TSAP-spezifischem Engpaß
T_RSAPNOTATT	258	Abbau vom Partner-CCP, weil der adressierte TSAP dort nicht angemeldet ist
T_RUNSAP	259	Abbau vom Partner-CCP, weil der adressierte TSAP dort nicht bekannt ist
T_RCONGEST	385	Abbau vom Partner-CCP wegen Betriebsmittelengpaß
T_RCONNFAIL	386	Abbau vom Partner-CCP wegen Mißlingen des Verbindungsaufbaus, z.B. weil Benutzerdaten zu lang oder Vorrangdaten nicht zugelassen
T_RDUPREF	387	Abbau vom Partner-CCP, weil für ein NSAP-Paar eine zweite Verbindungsreferenz vergeben wurde (Systemfehler)
T_RMISREF	388	Abbau vom Partner-CCP wegen einer nicht zuzuordnenden Verbindungsreferenz (Systemfehler)
T_RPROTERR	389	Abbau vom Partner-CCP wegen Protokollfehler (Systemfehler)
T_RREFOFLOW	391	Abbau vom Partner-CCP wegen Verbindungsreferenz-Überlauf
T_RNOCONN	392	Aufbau der Netzverbindung vom Partner-CCP abgelehnt
T_RINLNG	394	Abbau vom Partner-CCP wegen falscher Header- oder Parameterlänge (Systemfehler)
T_RLCONGEST	448	Abbau vom lokalen CCP wegen Betriebsmittelengpaß
T_RLNOQOS	449	Abbau vom lokalen CCP, weil Quality of Service nicht mehr geboten werden kann
T_RLPROTERR	464	Abbau vom lokalen CCP wegen Transportprotokollfehler (Systemfehler)
T_RLINTIDU	465	Abbau vom lokalen CCP, weil es eine zu lange Schnittstellen-Dateneinheit (TIDU) erhalten hat (Systemfehler)
T_RLNORMFLOW	466	Abbau vom lokalen CCP wegen Verletzung der Flußregelungsvorschriften für Normaldaten (Systemfehler)

T_RLEXFLOW	467	Abbau vom lokalen CCP wegen Verletzung der Flußregelungsvorschriften für Vorrangdaten (Systemfehler)
T_RLINSAPID	468	Abbau vom lokalen CCP, weil es eine ungültige TSAP-Id. erhalten hat (Systemfehler)
T_RLINCEPID	469	Abbau vom lokalen CCP, weil es eine ungültige TCEP-Id. erhalten hat (Systemfehler)
T_RLINPAR	470	Abbau vom lokalen CCP wegen eines unzulässigen Parameterwerts, z.B. Benutzerdaten zu lang oder Vorrangdaten nicht zugelassen
T_RLNOPERM	480	Aufbau durch Administration des lokalen CCP verhindert
T_RLPERMLOST	481	Abbau durch Administration des lokalen CCP
T_RLNOCONN	482	Aufbau vom lokalen CCP nicht durchführbar, weil keine Netzverbindung verfügbar ist
T_RLCONNLOST	483	Abbau vom lokalen CCP wegen Verlust der Netzverbindung
T_RLNORESP	484	Aufbau vom lokalen CCP nicht durchführbar, weil Partner nicht auf CONRQ antwortet
T_RLIDLETRAF	485	Abbau vom lokalen CCP wegen Verlust der Verbindung (Idle Traffic Timeout)
T_RLRESYNC	486	Abbau vom lokalen CCP, weil Resynchronisierung erfolglos war (mehr als 10 Wdh.)
T_RLEXLOST	487	Abbau vom lokalen CCP, weil Vorrangdatenkanal defekt ist (mehr als 3 Wdh.)

Diese Gründe treten nur bei Stationskopplung auf:

T_SYS4	4	Partner antwortet nicht innerhalb des Zeitlimits
T_SYS8	8	Partner nicht verfügbar
T_SYS12	12	Verbindungsparameter nicht akzeptiert
T_SYS20	20	Betriebsmittelengpaß
T_SYS28	28	Systemabschaltung
T_SYS32	32	Systemfehler
T_SYS56	56	Abbauanforderung von Systemadministration
T_SYS64	64	Partner unbekannt
T_SYS255	255	Systemfehler

NAME

t_disrq - Verbindung abbauen, disconnection request

SYNOPSIS

```
#include <cmx.h>

int t_disrq(tref, opt)
int *tref;
union {struct t_optc2 optc2;} *opt;
```

BESCHREIBUNG

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. Aber auch CMX kann die Verbindung abbauen, wenn interne Umstände es erfordern. Der Aufruf von **t_disrq()** kann Dateneinheiten, die noch unterwegs sind, überholen. Diese gehen dann verloren. Mit dem Verbindungsabbau können der fernen TS-Anwendung Benutzerdaten mitgeschickt werden, falls das Transportsystem diese Option bietet.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

-> opt
Zeiger zu einer Variante mit Systemoptionen oder die Angabe NULL. Damit werden die Benutzerdaten angegeben, die die ferne TS-Anwendung mit der Anzeige des Verbindungsabbaus erhalten soll. Bei Angabe NULL nimmt CMX die 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: undefiniert

t_udatal
Länge der aus dem Bereich t_udatap zu übertragenden Benutzerdaten.
Standardwert: 0

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich.

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

NAME

t_error - Fehlerdiagnose

SYNOPSIS

```
#include <cmx.h>

int t_error()
```

BESCHREIBUNG

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

RÜCKMELDUNG

T_NOERROR
Kein Fehler.

T_UNSPECIFIED
Nicht näher spezifizierter Fehler.

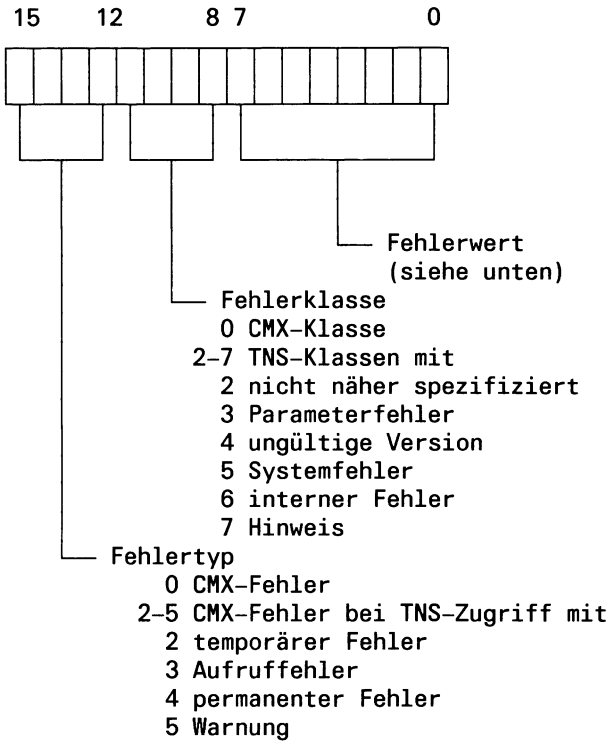
Diese beiden Fehlertypen sind systemunspezifisch.

sonst: systemspezifischer Fehlertyp, ist in cmx.h definiert und im folgenden kommentiert.

Überblick

Die Ausführungen beziehen sich auf CMX V2.1A. Die möglichen Meldungen zu den Aufrufen an ICMX(L) entstehen entweder in der CMX-Bibliothek im Benutzerprozeß oder im Betriebssystemkern. Die letzteren müssen danach unterschieden werden, ob sie in CMX selbst erzeugt werden oder aus Betriebssystemaufrufen in CMX resultieren.

Die Fehlermeldung wird in 16-Bit dargestellt:



Sie ist von links her auszuwerten. Weiter unten sind die Fehlerwerte zu den einzelnen Funktionen von ICMX(L) aufgeführt.

CMX-Fehlerwerte

Die von der CMX erzeugten Fehlerwerte sind die folgenden:

0	T_NOERROR	Kein Fehler
2	T_ENOENT	alle intern bereitgestellten Resourcen belegt
5	T_EIO	CCP-Ausfall
9	T_EBADF	Funktionsaufruf in diesem Zustand unzulässig
12	T_ENOMEM	Arbeitsspeicher unzureichend
14	T_EFAULT	Unzulässige Adresse
22	T_EINVAL	Unzulässiges Argument
100	T_UNSPECIFIED	nicht näher spezifierter Fehler
101	T_WSEQUENCE	Funktionsaufruf in diesem Zustand unzulässig
102	T_WREQUEST	unzulässiger Funktionsaufruf
103	T_WPARAMETER	fehlerhafter Parameter
104	T_WAPPLICATION	unbekannte Anwendung bzw. Anwendung bereits unter diesem Namen bekannt
105	T_WAPP_LIMIT	keine Anmeldung von Anwendungen mehr möglich
106	T_WCONN_LIMIT	Grenzwert für Verbindungen erreicht
107	T_WTREF	unzulässige Transportreferenz
108	T_WTUI_MSG	Fehlerhafte TUI-Netzmeldung (nur bei Stationskopplung, falsch generiert?)
109	T_COLLISION	Kollision aktiver/passiver Verbindungsaufbau bzw. -umlenkung oder Kollision des Aufrufes mit vorliegendem T_DISIN
110	T_WPROC_LIMIT	zu viele Prozesse haben Anwendungen angemeldet
111	T_NOCCP	Kein CCP für gewünschte Anwendung oder Verbindung vorhanden
112	T_ETIMEOUT	CCP reagiert nicht rechtzeitig
113	T_WROUTINFO	Unzulässige Routing Information (CC-Index)
114	T_CCP_END	CCP ist nicht mehr betriebsbereit
115	T_WRED_LIMIT	zu viele Verbindungsumlenkungen (temporär)
116	T_WLIBVERSION	Version der verwendeten CMX-Bibliothek wird nicht unterstützt

Ist der Fehlerwert keiner der obigen, so resultiert er aus einem in CMX verwendeten SINIX-Systemaufruf. Er ist dann in errno.h definiert [CES].

CCP-spezifische Meldungen

CCP-spezifische Meldungen gibt es nur bei Stationskopplung. Sie stammen aus einer der folgenden Klassen:
Meldungen für CCP-STA1 (bis V2.0),
Meldungen für CCP-STA2.

Verschiedene dieser Werte sind gleich, haben aber je nach Anschluß unterschiedliche Interpretation.

CCP-STA1:

```
48 T_EPARUOPEN  Ein Kanal ist eröffnet,
                keine Parametrisierung möglich
49 T_EFPARAM    Parameter-Fehler, Standleitung,
                halbduplex nicht zulässig
50 T_EPARAM     keine Parameter-Werte vorhanden
51 T_EMODFL     Modem-Fehler, M1 ohne S1 gesetzt
52 T_ECTS       CTS/M2 nicht gesetzt innerhalb 2 sec,
                Meldung erfolgt alle 2 sec
53 T_EDSR       DSR/M1 nicht mehr gesetzt, Prozedurabbruch
54 T_EDLEEOT    Prozedurabbruch, DLE EOT empfangen,
                nur bei einer Wählleitung
55 T_EWTOUT24   kein Steuerzeichen-Empfang innerhalb 24 s
                bei Wählleitung, Prozedurabbruch
56 T_ESTOUT24   kein Steuerzeichen-Empfang innerhalb 24 s
                bei Standleitung
57 T_EWRCOUNT   Anzahl der auszusendenden Daten zu groß
58 T_EWRNGC     auszusendende Daten enthalten Prozedursteuerzeichen
64 T_EPRRWZ     MSV1-Prozedurfehler beim Empfang, Text
                falsch empfangen, Wiederholzähler abgelaufen
65 T_EPRREOT    MSV1-Prozedurfehler beim Empfang,
                unerwartetes EOT nach STX Empfang
66 T_EPRXWZ     MSV1-Prozedurfehler beim Senden, Text
                konnte nicht gesendet werden,
                Wiederholzähler abgelaufen
67 T_EPRXWABT   MSV1-Prozedurfehler beim Senden, Text gut
                gesendet, aber WABT-Zähler abgelaufen
80 T_EPROZEMPF  jetzt wieder Prozedur-Zeichen empfangen
```

CCP-STA2:

```
48 T_EPARUOPEN  Ein Kanal ist eröffnet,
                keine Parametrisierung möglich
50 T_EPARAM     keine Leitungsparameter vorhanden
51 T_EMODFL     Open: (HW)-Error
52 T_ECTS       Modem oder HW-Fehler
53 T_EDSR       Prozedurabbruch (Ltg nicht mehr vorhanden?)
56 T_ESTOUT24   Polling ausgefallen
57 T_EWRCOUNT   Anzahl der Sende-/Empfangsdaten falsch
64 T_EPRRWZ     Daten konnten nicht empfangen werden
66 T_EPRXWZ     Daten konnten nicht gesendet werden
68 T_EDATLST    Datenverlust beim Empfang
80 T_EPROZEMPF  Polling wieder registriert
```

Fehlermeldungen zu CMX-Aufrufen

Im folgenden sind die CMX-Aufrufe in alphabetischer Reihenfolge mit den möglichen Fehlerwerten bei Fehlertyp "CMX-Fehler" (bei gewissen Aufrufen auch zum Fehlertyp "CMX-Fehler bei TNS-Zugriff") zusammengestellt. Es sind nur die Meldungen aufgeführt, die in CMX explizit erzeugt werden. Meldungen des CCP oder Meldungen zu SINIX-Systemaufrufen sind nicht eingeschlossen. Zur Darstellung: bei nachgestelltem "(STA)" gilt die Ursache nur bei Stationskopplung. Bei nachgestelltem "(INT)" handelt es sich um einen CMX-internen Fehler.

t_attach(name, opt)

```
  2 T_ENOENT
    * keine Gerätedatei frei (zu viele Anmeldungen)
 14 T_EFAULT
    * name oder opt (!= NULL) sind ungültige Adressen
 22 T_EINVAL
    * name unbekannt (Konfigurierung fehlt?) (STA)
    * unbekanntes ioctl-Argument (INT)
 50 T_EPARAM
    * CMX und/oder CCP noch nicht konfiguriert (STA)
103 T_WPARAMETER
    * LOKALER NAME in name falsch
      t_mnmode != T_MNMODE
      t_mnlng <= 0 oder > sizeof(struct t_myname)
      LOKALER NAME nicht für Stationskopplung geeignet (STA)
      t_mnlng zu groß (STA)
      cpn_selsize <= 0
      cps_lng <= 0 oder > 10 für mind. einen T-Selektor
    * opt (!= NULL) falsch
      t_optnr != T_OPTA1
      t_apmode enthält weder T_PASSIVE noch T_ACTIVE
      noch T_REDIRECT
      t_conlim <= 0 oder > CONLIMIT
104 T_WAPPLICATION
    * Prozeß bereits unter name angemeldet
105 T_WAPP_LIMIT
    * Prozeß bereits in zu vielen Anwendungen angemeldet (STA)
    * zuviele Anwendungen insgesamt angemeldet
    * es bestehen zu viele Anmeldungen insgesamt
110 T_WPROC_LIMIT
    * zuviele Prozesse in Anwendungen angemeldet
111 T_NOCCP
    * name konnte bei keinem CCP erfolgreich angemeldet werden
```

t_concf(tref, opt)

```
  5 T_EIO
 12 T_ENOMEM
 14 T_EFAULT
    * opt (!= NULL) ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * T_CONCF wurde nicht angezeigt
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * opt (!= NULL) falsch
      t_optnr != T_OPTC1
      t_udatal zu klein
107 T_WTREF
108 T_WTUI_MSG
112 T_ETIMEOUT
114 T_CCP_END
```

t_conin(tref, toaddr, fromaddr, opt)

```
  5 T_EIO
 14 T_EFAULT
    * toaddr oder fromaddr oder opt (!= NULL) ist ungültige-
Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * T_CONIN liegt nicht vor
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * opt (!= NULL) falsch
      t_optnr != T_OPTC1
      t_udatal zu klein
107 T_WTREF
108 T_WTUI_MSG
114 T_CCP_END
```

t_conrq(tref, toaddr, fromaddr, opt)

```
  5 T_EIO
  9 T_EBADF
    * aktiver Verbindungsaufbau nicht zulässig (STA)
 14 T_EFAULT
    * toaddr oder fromaddr oder opt (!= NULL) ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * aktiver Verbindungsaufbau nicht zulässig (t_apmode)
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * LOKALER NAME in fromaddr falsch:
      t_mnmode != T_MNMODE
      LOKALER NAME für Stationskopplung falsch (STA)
      t_mnlng falsch (STA)
    * TRANSPORTADRESSE in toaddr falsch:
      t_pamode != T_PAMODE
      t_palng falsch
      TRANSPORTADRESSE für Stationskopplung falsch (STA)
      sa_pname enthält falsch formatierten Namen (STA)
      Rechnernummer mehr als 2 Zeichen,
      Regionsnummer mehr als 3 Zeichen,
      '/' zwischen beiden fehlt,
      sa_pname nicht mit ' ' aufgefüllt (STA)
    * opt (!= NULL) falsch
            t_optnr != T_OPTC1 (STA)
            t_udadal < 0 oder > T_MSG_SIZE
            t_udatal falsch (STA)
            t_xdata != T_NO (STA)
            t_timeout < 0
104 T_WAPPLICATION
    * Prozeß nicht mit fromaddr angemeldet
106 T_WCONN_LIMIT
    * Prozeß hat bereits t_conlim Verbindungen
    * alle möglichen Verbindungen des PC bereits aufgebaut
108 T_WTUI_MSG
109 T_COLLISION
    * Verbindungsauf oder Verbindungsumlenkung bereits angezeigt
111 T_NOCCP
    * CCP für toaddr ist nicht aktiv
    * keine Anmeldung für CCP gemäß toaddr
112 T_ETIMEOUT

113 T_WROUTINFO
114 T_CCP_END
```

t_conrs(tref, opt)

```
  5 T_EIO
 12 T_ENOMEM
 14 T_EFAULT
    * opt (!= NULL) ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * kein erfolgreicher t_conin vorausgegangen
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * opt (!= NULL) falsch
      t_optnr != T_OPTC1
      t_udatal < 0 oder > T_MSG_SIZE
      t_udatal != 0 (STA)
      t_xdata != T_NO (STA)
      t_timeout < 0
107 T_WTREF
112 T_ETIMEOUT
114 T_CCP_END
```

t_datago(tref)

```
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * nicht in Datenaustauschphase
    * T_REDIN für diese Verbindung nicht mit t_redin abgeholt
    * Datenanzeige ist nicht gesperrt
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
107 T_WTREF
```

t_datain(tref, datap, datal, chain)

```
  5 T_EIO
  9 T_EBADF
    * T_DATAIN liegt nicht vor (INT)
 14 T_EFAULT
    * datap ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * T_DATAIN liegt nicht vor
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * datal > TIDU-Länge (STA)
    * datal zu klein (INT)
107 T_WTREF
114 T_CCP_END
```

t_datarq(tref, datap, datal, chain)

```
  5 T_EIO
 14 T_EFAULT
    * datap ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * nicht in Datenaustauschphase
    * T_REDIN für diese Verbindung nicht mit t_redin abgeholt
    * Datenfluß ist gesperrt
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * datal < 1 oder datal > TIDU-Länge
    * chain != T_MORE und chain != T_END
    * chain != T_END (STA)
107 T_WTREF
112 T_ETIMEOUT
114 T_CCP_END
```

t_datastop(tref)

```
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * nicht in Datenaustauschphase
    * T_REDIN für diese Verbindung nicht mit t_redin abgeholt
    * Datenanzeige ist schon gesperrt
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
    * eine angezeigte Dateneinheit oder Vorrangdateneinheit
      ist noch nicht vollstandig gelesen
107 T_WTREF
```

t_detach(name)

```
 14 T_EFAULT
    * name ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * name (= LOKALER NAME) falsch
          t_mnmode != T_MNMODE
          t_mnlng <= 0 oder > sizeof(struct t_myname)
          name nicht für Stationskopplung geeignet (STA)
          t_mnlng zu groß (STA)
104 T_WAPPLICATION
    * Prozeß nicht mit name angemeldet
```

t_disin(tref, reason, opt)

```
  5 T_EIO
 14 T_EFAULT
    * opt (!= NULL) ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * T_DISIN liegt nicht vor
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * opt (!= NULL) falsch
          t_optnr != T_OPTC2
          t_udatal zu klein
    * opt != NULL (STA)
107 T_WTREF
108 T_WTUI_MSG
114 T_CCP_END
```

t_disrq(tref, opt)

```
 14 T_EFAULT
    * opt (!= NULL) ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
    * Verbindung nicht vorhanden oder gerade im aktivem Aufbau
103 T_WPARAMETER
    * opt (!= NULL) falsch
          t_optnr != T_OPTC2
          t_udatal < 0 oder > T_MSG_SIZE
    * opt != NULL (STA)
107 T_WTREF
108 T_WTUI_MSG
```

t_event(tref, cmode, opt)

```
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
    * eine angezeigte Dateneinheit oder Vorrangdateneinheit
      ist noch nicht vollstandig gelesen
103 T_WPARAMETER
    * opt != NULL
    * cmode != T_CHECK und cmode != T_WAIT
108 T_WTUI_MSG
```

t_info(tref, opt)

```
 14 T_EFAULT
    * opt (!= NULL) ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * opt == NULL oder opt->t_optnr ungleich T_OPTI1
107 T_WTREF
```

t_redin(tref, pid, opt)

```
  9 T_EBADF
    * T_REDIN liegt nicht vor
 12 T_ENOMEM
 14 T_EFAULT
    * opt (!= NULL) ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
101 T_WSEQUENCE
    * T_REDIN liegt nicht vor
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * opt (!= NULL) falsch
    t_optnr != T_OPTC2
    t_udatal < T_RED_SIZE
107 T_WTREF
```

t_redrq(tref, pid, opt)

```
  2 T_ENOENT
    * alle zugänglichen Betriebsmittel ausgeschöpft
  9 T_EBADF
    * Prozeß hält keine Verbindung (INT)
 12 T_ENOMEM
    * Maximalzahl gleichzeitiger, noch nicht
      abgeschlossener Verbindungsumlenkungen erreicht
 14 T_EFAULT
    * opt (!= NULL) ist ungültige Adresse
 22 T_EINVAL
    * unbekanntes ioctl-Argument (INT)
    * Zielprozeß für Umlenkung nicht gefunden
```

```
101 T_WSEQUENCE
    * nicht in Datenaustauschphase
    * Datenfluß ist gesperrt
    * eine angezeigte Dateneinheit oder Vorrangdateneinheit
      ist noch nicht vollstandig gelesen
    * für diese Verbindung angezeigtes T_REDIN noch nicht
      mit t_redin abgeholt
102 T_WREQUEST
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * pid ist eigene pid
    * pid ist nicht in dieser Anwendung
    * pid kann kein T_REDIRECT entgegennehmen (t_apmode)
    * opt (!= NULL) falsch
      t_optnr != T_OPTC2
      t_udatal > T_RED_SIZE
      t_udatal < 0
106 T_WCONN_LIMIT
    * pid hat bereits t_conlim Verbindungen
107 T_WTREF
115 T_WRED_LIMIT
```

t_xdatgo(tref)

```
101 T_WSEQUENCE
    * nicht in Datenaustauschphase
    * T_REDIN für diese Verbindung nicht mit t_redin abgeholt
    * Vorrangdatenanzeige ist nicht gesperrt
102 T_WREQUEST
    * Option nicht verfügbar (STA)
    * Prozeß ist in keiner Anwendung angemeldet
107 T_WTREF
```

t_xdatin (tref, datap, datal)

```
  5 T_EIO
 14 T_EFAULT
    * datap ist ungültige Adresse
101 T_WSEQUENCE
    * T_XDATIN liegt nicht vor
102 T_WREQUEST
    * Option nicht verfügbar (STA)
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * datal zu klein (INT)
107 T_WTREF
114 T_CCP_END
```

t_xdatrq(tref, datap, datal)

```
  5 T_EIO
 14 T_EFAULT
    * datap ist ungültige Adresse
101 T_WSEQUENCE
    * nicht in Datenaustauschphase
    * T_REDIN für diese Verbindung nicht mit t_redin abgeholt
    * Datenfluß ist gesperrt
102 T_WREQUEST
    * Option nicht verfügbar (STA)
    * Prozeß ist in keiner Anwendung angemeldet
103 T_WPARAMETER
    * datal < 1 oder datal > T_EXP_SIZE
107 T_WTREF
112 T_ETIMEOUT
114 T_CCP_END
```

t_xdatstop(tref)

```
101 T_WSEQUENCE
    * nicht in Datenaustauschphase
    * T_REDIN für diese Verbindung nicht mit t_redin abgeholt
    * Datenanzeige ist schon gesperrt
102 T_WREQUEST
    * Option nicht verfügbar (STA)
    * Prozeß ist in keiner Anwendung angemeldet
    * eine angezeigte Dateneinheit oder Vorrangdateneinheit
      ist noch nicht vollständig gelesen
107 T_WTREF
```

SIEHE AUCH

cmx.h - globale ICMX-Definitionsdatei
errno.h - Meldungen zu SINIX-Systemaufrufen

NAME

t_event - Ereignis abwarten oder abfragen, event

SYNOPSIS

```
#include <cmx.h>

int t_event(tref, cmode, opt)
int *tref;
int cmode;
union {} *opt;
```

BESCHREIBUNG

t_event() wartet entweder darauf, daß ein CMX-Ereignis eintritt oder prüft nach, ob ein CMX-Ereignis inzwischen eingetreten ist. Dies wird durch den Parameter **cmode** gesteuert. Liegen mehrere Ereignisse für eine Verbindung vor, werden sie nacheinander in der Reihenfolge angezeigt, wie sie aufgetreten sind. Allerdings kann das Ereignis T_DISIN Dateneinheiten für die zugehörige Verbindung überholen und damit zerstören; das Ereignis T_XDATIN kann Dateneinheiten, ohne sie zu zerstören, überholen. **t_event()** gibt CMX außerdem die Erlaubnis, die nächste Dateneinheit für eine Verbindung abzusetzen, wenn nicht der Datenfluß für diese Verbindung explizit durch **t_datastop()** bzw. **t_xdatstop()** gesperrt wurde.

Parameter:

<- tref
Zeiger zu einem Feld, in dem CMX die verbindungsspezifische Transportreferenz zurückliefert. Sie gibt an, zu welcher Verbindung das Ereignis gehört. Nach T_NOEVENT oder T_ERROR ist der Inhalt von **tref** nicht definiert.

-> cmode
gibt an, ob **t_event()** auf ein Ereignis warten soll oder nicht.
Mögliche Werte:

T_WAIT
t_event() wartet auf das nächste eintretende CMX-Ereignis (synchrone Verarbeitung), der Prozeß wird suspendiert, bis ein Ereignis eintritt. Mit einem Signal läßt sich der Aufruf abbrechen, er wird dann mit dem Ereignis T_NOEVENT beendet. Man kann z.B. die Funktion **alarm()** verwenden, um **t_event()** zeitlich zu überwachen.

T_CHECK
t_event() prüft, ob ein Ereignis ansteht (asynchrone Verarbeitung). Wenn nicht, kehrt **t_event()** mit T_NOEVENT zurück.

-> opt
Hier muß NULL angegeben werden, da in dieser Version keine Optionen unterstützt werden.

RÜCKMELDUNG

T_NOEVENT
falls cmode = T_CHECK: kein Ereignis da
falls cmode = T_WAIT: Abbruch, z.B. durch Signal

Der Inhalt von **tref** ist nicht definiert.

Hinweis
Das Ereignis T_NOEVENT führt die Zustandsautomaten für den Empfang von Normal- und Vorrangdaten aus dem Zustand D1 in D3 über (siehe t_intro). D3 kann nur beim Ereignis T_DATAIN bzw. T_XDATIN verlassen werden.

T_DATAIN
Empfangs-Anzeige für Daten, muß mit **t_datain()** abgeholt werden.

Hinweis
CMX zeigt dieses Ereignis für eine Verbindung nicht an, solange auf ihr der Datenfluß gesperrt ist, d.h. wenn der empfangende Prozeß für sie **t_datastop()** gegeben hat.

T_DATAGO
Anzeige von CMX, daß die lokale TS-Anwendung auf der in **tref** angegebenen Verbindung wieder Daten senden darf. Mögliche Reaktion: **t_datarq()**.

Hinweis
Dieses Ereignis impliziert, daß auch wieder Vorrangdaten gesendet werden können, sofern beim Verbindungsaufbau das Senden und Empfangen von Vorrangdaten vereinbart wurde.

T_XDATIN
Empfangs-Anzeige für Vorrangdaten, muß mit **t_xdatin()** abgeholt werden.

Hinweis
- Die Anzeige dieses optionalen Ereignisses ist nur möglich, wenn beim Verbindungsaufbau das Senden und Empfangen von Vorrangdaten vereinbart wurde.
- CMX zeigt dieses Ereignis nicht an, solange der Vorrangdatenfluß gesperrt ist, d.h wenn der empfangende Prozeß **t_xdatstop** gegeben hat.

T_XDATGO
Anzeige von CMX, daß der Prozeß auf der in **tref** angegebenen Verbindung wieder Vorrangdaten senden darf.
Mögliche Reaktion: **t_xdatrq()**.

Hinweis
Die Anzeige dieses optionalen Ereignisses ist nur möglich, wenn beim Verbindungsaufbau das Senden und Empfangen von Vorrangdaten vereinbart wurde.

T_CONIN
Verbindungsaufbauanzeige, muß mit **t_conin()** abgeholt, dann mit **t_conrs()** bestätigt oder mit **t_disrq()** abgelehnt werden.

T_CONCF
Verbindungsaufbaubestätigung von der gerufenen TS-Anwendung, muß mit **t_concf()** abgeholt werden.

T_DISIN
Verbindungsabbauanzeige, muß mit **t_disin()** abgeholt werden.

T_REDIN
Verbindungsumlenkungsanzeige, muß mit **t_redin()** abgeholt werden.

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

NAME

t_info - Informationen über CMX, information

SYNOPSIS

```
#include <cmx.h>

int t_info(tref, opt)
int *tref;
union {struct t_opti1 opti1;} *opt;
```

BESCHREIBUNG

t_info() informiert über Konstante in CMX. In Version 2 ist als einzige Konstante die Länge der Dateneinheit für die angegebene Verbindung definiert. Diese Größe hängt vom verwendeten Transportsystem ab.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

<- 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
Feld, in das CMX die maximale Länge einer zu übertragenden Dateneinheit einträgt. Dieser Wert gibt an, wieviele Byte man maximal beim Datentransfer über diese Verbindung senden bzw. empfangen kann.

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich.

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

NAME

t_redin - Umgelenkte Verbindung annehmen, redirection indication

SYNOPSIS

```
#include <cmx.h>

int t_redin(tref, pid, opt)
int *tref;
int *pid;
union {struct t_optc2 optc2;} *opt;
```

BESCHREIBUNG

t_redin() nimmt ein zuvor mit **t_event()** gemeldetes Ereignis T_REDIN entgegen. T_REDIN zeigt an, daß ein anderer Prozeß derselben TS-Anwendung eine Verbindung auf den laufenden Prozeß umgelenkt hat. Das Ereignis T_REDIN muß mit **t_redin()** entgegengenommen werden. Ist die Verbindung unerwünscht, kann sie mit **t_redrq()** dem ursprüglichen Prozeß zurückgeben oder mit **t_disrq()** abgebaut werden. Der Aufruf liefert die Prozeßidentifikation des rufenden Prozesses sowie Benutzerdaten, die er bei der Umlenkung mitgeschickt hat.

Parameter:

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

<- pid
Zeiger zu einem Feld, in dem CMX die Prozeßidentifikation des rufenden Prozesses zurückliefert.

<> opt
Zeiger zu einer Variante mit Systemoptionen oder die Angabe NULL. Damit werden die Benutzerdaten entgegengenommen, die der rufende Prozeß bei der Verbindungsumlenkanforderung mitgegeben hat. Bei Angabe NULL wirft CMX die Benutzerdaten weg. Hat der rufende Prozeß nichts angegeben, setzt CMX die 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 auf einen Datenbereich, in den CMX die empfangenen Benutzerdaten des rufenden Prozesses einträgt.

t_udatal
: Vor dem Aufruf muß hier die Länge des Datenbereiches t_udatap stehen, sie muß mindestens T_RED_SIZE betragen. CMX liefert in diesem Feld die Anzahl der empfangenen Byte zurück. Standardwert: 0.

RÜCKMELDUNG

T_OK
: Aufruf war erfolgreich.

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

NAME

t_redrq - Verbindung umlenken, redirection request

SYNOPSIS

```
#include <cmx.h>

int t_redrq(tref, pid, opt)
int *tref;
int *pid;
union {struct t_optc2 optc2;} *opt;
```

BESCHREIBUNG

t_redrq() gibt die angegebene Verbindung an den angegebenen anderen Prozeß derselben TS-Anwendung weiter. Die Verbindung ist dann im rufenden Prozeß nicht mehr bekannt, die Transportreferenz für diesen Prozeß ungültig. Der gerufene Prozeß erhält das Ereignis T_REDIN. Die Verbindung darf nicht umgelenkt werden, wenn für sie T_DATASTOP oder T_XDATSTOP vorliegt, während eine Dateneinheit dieser Verbindung stückweise mit **t_datain()** (Ergebnis > 0) abgeholt wird oder wenn der jüngste **t_event()** mit Ergebnis T_NOEVENT zurückkehrte. Mit der Verbindungsumlenkung können dem empfangenden Prozeß Benutzerdaten mitgegeben werden.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

-> pid
Zeiger zu einem Feld, in dem die Prozeßidentifikation des gerufenen Prozesses anzugeben ist.

-> opt
Zeiger zu einer Variante mit Benutzeroptionen oder Angabe NULL. Damit können Sie dem gerufenen Prozeß bei der Verbindungsumlenkung Informationen mitschicken. Er nimmt diese mit der Verbindungsumlenkung entgegen. Wählen Sie den NULL-Zeiger, so stellt CMX dem gerufenen Prozeß die Standardwerte zu. 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 CMX dem empfangenden Prozeß zustellen soll.

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.
Standardwert: 0

RÜCKMELDUNG

T_OK
: Aufruf war erfolgreich.

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

NAME

t_xdatgo - Vorrangdatenfluß freigeben, expedited datago

SYNOPSIS

```
#include <cmx.h>

int t_xdatgo(tref)
int *tref;
```

BESCHREIBUNG

t_xdatgo() gibt den gesperrten Vorrangdatenfluß auf der angegebenen Verbindung frei, sofern beim Aufbau dieser Verbindung mit der sendenden TS-Anwendung deren Austausch vereinbart wurde. Der Prozeß teilt CMX damit mit, daß er wieder bereit ist, Vorrangdaten zu empfangen. Der Aufruf bewirkt im einzelnen:

- Der laufende Prozeß erhält wieder das Ereignis T_XDATIN für die angegebene Verbindung zugestellt, falls es ansteht.
- Die sendende TS-Anwendung erhält das Ereignis T_XDATGO zugestellt. Sie darf wieder Vorrangdaten senden. Normaldaten sind von t_xdatgo nicht betroffen.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich.

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

NAME

t_xdatin - Vorrangaten empfangen, expedited data indication

SYNOPSIS

```
#include <cmx.h>

int t_xdatin(tref, datap, datal)
int *tref;
char *datap;
int *datal;
```

BESCHREIBUNG

t_xdatin() nimmt ein zuvor mit **t_event()** gemeldetes Ereignis T_XDATIN entgegen. T_XDATIN zeigt an, daß auf der angegebenen Verbindung Vorrangdaten der sendenden TS-Anwendung eingetroffen sind. Vorraussetzung ist, daß beim Aufbau der Verbindung mit der sendenden TS-Anwendung deren Austausch vereinbart wurde. Wenn die Vorrangdaten in den bereitgestellten Bereich **datap** hineinpassen, kehrt der Aufruf mit T_OK zurück. Passen die Vorrangdaten nicht in den bereitgestellten Bereich, liefert der Aufruf einen Wert $n > 0$. Dies ist die Anzahl der Byte, die von den Vorrangdaten noch nicht gelesen wurden (Restlänge). In diesem Fall muß **t_xdatin()** solange aufgerufen werden, bis alle Vorrangdaten gelesen sind. Erst dann können wieder andere CMX-Aufrufe, z.B. **t_event()** abgesetzt werden. Die maximale Länge der Vorrangdaten hängt ab vom verwendeten Transportsystem, ist aber nie größer als T_EXP_SIZE.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

<- datap
Zeiger auf einen Bereich, in den CMX die empfangenen Vorrangdaten einträgt.

< > datal
Vor dem Aufruf muß hier die Länge des Datenbereiches datap stehen (mindestens 1). CMX liefert in diesem Feld die Anzahl der eingetragenen Byte zurück.

RÜCKMELDUNG

T_OK
Aufruf erfolgreich, Vorrangdaten vollständig gelesen.

n > 0
Es sind noch n Byte in den Vorrangdaten explizit vorhanden. Sie müssen Sie erst mit weiteren Aufrufen t_xdatin() abholen, bevor Sie mit **t_event** das nächste Ereignis prüfen können.

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

NAME

t_xdatrq - Vorrangdaten senden, expedited data request

SYNOPSIS

```
#include <cmx.h>

int t_xdatrq(tref, datap, datal)
int *tref;
char *datap;
int *datal;
```

BESCHREIBUNG

t_xdatrq() sendet auf der angegebenen Verbindung Vorrangdaten an die empfangende TS-Anwendung, sofern beim Aufbau dieser Verbindung mit der empfangenden TS-Anwendung deren Austausch vereinbart wurde. Die maximale Länge der Vorrangdaten hängt ab vom verwendeten Transportsystem, kann aber mehr als T_EXP_SIZE Byte nie überschreiten. Vorrangdaten unterliegen einer eigenen Flußregelung. Sie können normale Dateinheiten überholen. Das Transportsystem garantiert, daß Vorrangdaten nie später beim Partner eintreffen als danach abgeschickte Normaldateneinheiten. Bei Rückmeldung T_XDATSTOP sind die Vorrangdaten übernommen, der Vorrangdatenfluß aber für diese Verbindung gesperrt worden. Dies kann auf Initiative der empfangenden TS-Anwendung geschehen (mit **t_xdatstop()**) oder auf Initiative von CMX, wenn der lokale Zwischenspeicher voll ist. Es muß dann mit **t_event()** erst das Ereignis T_XDATGO oder T_DATAGO für diese Verbindung abgewartet werden, bevor weitere Vorrangdaten gesendet werden dürfen.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

-> datap
Zeiger zu einem Datenbereich mit den zu sendenden Vorrangdaten.

-> datal
Zeiger zu einem Feld, in das vor dem Aufruf die Länge von datap eintragen werden muß (mindestens 1). Die maximale Länge ist abhängig vom verwendeten Transportsystem, aber nie größer als T_EXP_SIZE.

RÜCKMELDUNG

T_OK
Aufruf erfolgreich, weitere Vorrangdaten können sofort gesendet werden.

T_XDATSTOP
Aufruf erfolgreich, weitere Vorrangdaten dürfen erst gesendet werden, wenn für diese Verbindung das Ereignis T_XDATGO oder T_DATAGO eingetroffen ist.

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

NAME

t_xdatstop - Vorrangdatenfluß sperren, expedited data stop

SYNOPSIS

```
#include <cmx.h>

int t_xdatstop(tref)
int *tref;
```

BESCHREIBUNG

t_xdatstop() sperrt den Vorrangdatenfluß und den Normaldatenfluß auf der angegebenen Verbindung, sofern Sie beim Aufbau dieser Verbindung mit der sendenden TS-Anwendung deren Austausch vereinbart haben. Der laufende Prozeß teilt CMX damit mit, daß er nicht bereit ist, Vorrangdaten oder Normaldaten zu empfangen. Ein bereits angezeigtes Ereignis T_XDATIN, T_DATAIN muß aber erst bearbeitet werden. Dieser Aufruf bewirkt, daß der laufende Prozess die Ereignisse T_XDATIN und T_DATAIN nicht mehr zugestellt bekommt. Er kann aber währenddessen andere CMX-Funktionen aufrufen, z.B. eine weitere Verbindung aufbauen. Die sendende TS-Anwendung erhält bei **t_xdatrq()** das Ergebnis T_XDATSTOP. Sie darf keine Daten oder Vorrangdaten mehr senden. Freigegeben wird der Vorrangdatenfluß mit **t_xdatgo()** oder mit t_datago(), falls auch der Normaldatenfluß freigegeben werden soll.

Parameter:

-> tref
Zeiger zu einem Feld mit der Transportreferenz.

RÜCKMELDUNG

T_OK
Aufruf war erfolgreich.

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

6.2 CMX-Aufrufe zum Transport Name Service

NAME

ICMX(TNS) - Programmschnittstelle zum Transport Name Service TNS in SINIX

BESCHREIBUNG

Die folgenden Seiten beschreiben die Programmschnittstelle ICMX(TNS) zum Transport Name Service (TNS) in SINIX. Sie enthalten

- einen Überblick über die Funktionen der Schnittstelle ICMX(TNS),
- die präzise Beschreibung der Funktionsausfrufe mit allen Parametern.

Der Transport Name Service in SINIX

Die Funktionen des TNS dienen zur Abfrage und Verwaltung konfigurationsspezifischer Eigenschaften zu TS-Anwendungen, wie TRANSPORT-ADRESSEN und LOKALE NAMEN. Der TNS bietet dazu eine benutzerfreundliche Namensgebung, mit der man sich auf TS-Anwendungen mit GLOBALEN NAMEN beziehen kann, die insbesondere für die Verwendung durch den Menschen geeignet sind. Der TNS übernimmt die erforderliche Abbildung GLOBALE NAMEN < > Eigenschaften, die die dynamische Zuordnung von Eigenschaften zu TS-Anwendungen erlaubt.

Der TNS stellt dem Benutzer auf Anforderung folgendes zur Verfügung:

Zuordnung von GLOBALEN NAMEN zu Eigenschaften:

womit eine Beziehung zwischen einem GLOBALEN NAMEN und einer Menge von Informationen über die TS-Anwendung, auf die sich der GLOBALE NAME bezieht, hergestellt wird.

Zuordnung von Eigenschaft zu GLOBALEN NAMEN:

womit die GLOBALEN NAMEN von solchen TS-Anwendungen aufgelistet werden können, die eine angegebene Eigenschaft besitzen.

ICMX(TNS) ist realisiert als eine Menge von C-Funktionen. Die TNS-Funktionen umfassen sowohl solche, mit denen man Informationen aus dem TS-Directory abfragen kann, als auch solche, mit denen man den Inhalt des TS-Directory verändern kann (letztere verwendet das Verwaltungsprogramm TNSADMIN [CCP] zur interaktiven und menügesteuerten Verwaltung des TS-Directory).

Einheitliche Namensstruktur von TS-Anwendungen

Ein GLOBALER NAME ist ein hierarchisch strukturierter Name, bestehend aus einer beliebigen Teilmenge von bis zu 5 Namensteilen (Namensteil 1 bis 5), mit von 1 nach 5 abfallender Hierarchie. Ein Namensteil wird beschrieben durch eine Identifikation, eine Länge und einen Wert, der nur aus abdruckbaren Zeichen des ISO-7-Bit-Codes bestehen darf. Der TNS macht keinerlei Vorgaben hinsichtlich der Bedeutung der Namensteile, außer der (strukturprägenden) hierarchischen Reihenfolge von 1 nach 5. Die folgenden Vereinbarungen entsprechen den Vorschlägen der internationalen Normungsgremien (die Definitionen T_... befinden sich in tnsx.h):

Namens-teil	Identi-fikation	Länge (dez*)	Bedeutung
1	TS_COUNTRY	TS_LCOUNTRY (2)	Country
2	TS_ADMD	TS_LADMD (16)	Administration Domain
3	TS_PRMD	TS_LPRMD (16)	Private Domain
4	TS_OU	TS_LOU (10)	Organizational Unit
5	TS_PN	TS_LPN (30)	Personal Name

*) gegenwärtige Maximallänge

Eigenschaften

Den GLOBALEN NAMEN kann eine Auswahl aus 6 Eigenschaften zugeordnet werden. Der TNS macht keine Vorgaben hinsichtlich interner Struktur oder Bedeutung der Eigenschaften. Die hier angegebene Interpretation entspricht ihrer Verwendung in CMX.

Eigenschaft	Bedeutung
TS_LNAME	LOKALER NAME einer TS-Anwendung, der sie in Ihrem lokalen Endsystem eindeutig kennzeichnet. Das Format hängt vom verwendeten Netzwerk ab.
TS_TRANS	TRANSPORTADRESSE einer TS-Anwendung, mit der sie im Netz ansprechbar ist. Das Format hängt vom verwendeten Netzwerk ab.
TS_ROUT	ROUTINGINFORMATION bei Mehrfachanschluß an ein und dasselbe Netzwerk.
TS_NEABX	MIGRATIONSSERVICE für TS-Anwendung, die mit alten Anwendungen in TRANSDATA kommunizieren wollen.
TS_GTYPE	GERAETETYP Unterscheidung einer gewöhnlichen von einer Emulationsanwendung. Nur für interne Migrationszwecke.
TS_TRSYS	TRANSPORTSYSTEM klassifiziert die Art des Netzwerkes, über das die TS-Anwendung erreichbar ist. Nur für Migrationszwecke erforderlich.

Eine Eigenschaft ist beschrieben durch Identifikation, Typ, Länge und Wert:

Identif.	Max. Länge	Wert
TS_LNAME	TS_LPROP	beliebig
TS_TRANS	TS_LPROP	beliebig
TS_ROUT	sizeof(short)	0...16
TS_NEABX	sizeof(char)	TS_TRUE|TS_FALSE
TS_GTYPE	sizeof(char)	TS_SS8110, TS_DSS9750 TS_DRS8122
TS_TRSYS	sizeof(char)	TS_NEA, TS_ISO, TS_LAN, TS_STA

Das TS-Directory

Der TNS verwaltet die Informationen über die TS-Anwendungen im Transport Service Directory (TS-Directory), einer Datenbasis, deren logische Struktur durch die oben erwähnte Namengebung geprägt ist. Das TS-Directory besteht aus Einträgen, von denen jeder zu einer TS-Anwendung gehört und über sie Informationen in Form von Eigenschaften enthält. Die Einträge sind durch den hierarchisch strukturierten GLOBALEN NAMEN der TS-Anwendung bezeichnet und daher im TS-Directory in einem Namensbaum (global naming tree) angeordnet. Der Baum besteht aus drei Elementen:

der Wurzel (ROOT),
den Knoten (NonLeafEntities),
den Blättern (LeafEntities).

Der Baum wächst von oben nach unten. Ein GLOBALER NAME entspricht einem Pfad im Namensbaum von der Wurzel zu einer LeafEntity oder einer NonLeafEntity, mit den Namensteilen als Pfadkomponenten. Alle Namensteile können Pfadkomponente zu einer LeafEntity sein, Namensteil 5 kann nicht Pfadkomponente zu einer NonLeafEntity sein. Die Festlegung NonLeafEntity oder LeafEntity erfolgt beim Einrichten des GLOBALEN NAMENS. An eine NonLeafEntity oder die Wurzel können unter Beachtung der Hierarchie der Namensteile eine weitere NonLeafEntity oder eine LeafEntity angefügt werden. Eigenschaften können nur den LeafEntities zugeordnet werden.

Der TNS unterstützt gleichzeitig bis zu 10 verschiedene TS-Directories mit den Identifikationen 0-9. Bei jeder TNS-Funktion muß als Parameter die Identifikation des TS-Directory angegeben werden, auf das sich die Funktion beziehen soll. Die Identifikation kann bei jedem Aufruf eine andere sein. Häufiges Umschalten zwischen den TS-Directories senkt aber die Performance des TNS, da in solchen Fällen die internen Caches zur Zugriffsoptimierung oft neu aufgebaut werden müssen.

CMX greift intern nur auf das TS-Directory TS_CMX_DIR_ID zu. Die TS-Directories sind im SINIX-Dateisystem als Dateiverzeichnisse /usr/lib/cmx/DIR<n> mit <n> = 0,...,9 abgelegt. Es ist möglich, dieses Dateiverzeichnis mit üblichen SINIX-Mitteln [SIN1] aufzulisten, die eingetragenen Dateien enthalten aber überwiegend binäre Information. Ein TS-Directory kann nicht mit den üblichen SINIX-Mitteln [SIN1] gelöscht oder auf einen anderen Rechner transportiert werden. Zu diesem Zweck gibt es eigene Funktionen im Verwaltungsprogramm TNSADMIN.

Filterung

Die TNS-Funktionen zur Informationsabfrage durchsuchen das TS-Directory nach GLOBALEN NAMEN, die bestimmten Vorgaben (Filterkriterien) genügen. Sie "filtern" aus der Gesamtheit aller GLOBALER NAMEN solche aus, die die Vorgaben erfüllen. Es gibt die zwei Filterkriterien "vorgegebene Namensteile" (die der GLOBALE NAME mindestens/genau haben muß, um den Filter zu passieren) und "vorgegebene Eigenschaft" (die der GLOBALE NAME haben muß, um den Filter zu passieren).

Bei Filterung kann die Ausgabe der Information in bereitgestellten Speicherbereichen oder in eine Datei erfolgen.

Speicherbestellung

Der Aufrufer einer TNS-Funktion muß im Programm ausreichend Speicherplatz für die vom TNS zu liefernde Information bereitstellen. Zeiger auf diese Speicherbereiche und Angaben zu ihrer Länge sind beim Aufruf in Parameterblöcken zu übergeben.

Funktionen des TNS

Die TNS-Funktionen bestehen aus zwei Gruppen von C-Funktionsaufrufen. Alle Funktionen haben als einziges Argument einen Zeiger auf einen Parameterblock, dessen Struktur in tnsx.h definiert ist.

Funktionsaufrufe zur Informationsabfrage	Beschreibung
ts11_read_leafs	Auflisten GLOBALER NAMEN mit vorgegebenen Namensteilen und Eigenschaften
ts12_read_children	Auflisten von Namensteilen der nächstniedrigeren Hierarchiestufe
ts13_read_properties	Abfrage der Eigenschaften zu einem GLOBALEN NAMEN

Funktionsaufrufe zur Informationsverwaltung	Beschreibung
ts21_create_non_leaf_entity	Einrichten einer NonLeafEntity im Namensbaum
ts22_delete_non_leaf_entity	Löschen einer NonLeafEntity im Namensbaum
ts23_create_leaf_entity	Einrichten einer LeafEntity im Namensbaum mit vorgegebenen Eigenschaften
ts24_delete_leaf_entity	Löschen einer LeafEntity im Namensbaum
ts25_add_property	Einrichten einer Eigenschaft an einer LeafEntity
ts26_delete_property	Löschen einer Eigenschaft an einer LeafEntity
ts27_change_property	Ändern einer Eigenschaft an einer LeafEntity

Fehlermeldungen zu den Aufrufen von ICMX(TNS)

Ein erfolgloser Funktionsaufruf von ICMX(TNS) oder ein erfolgreicher, verbunden mit einer Warnung an den Aufrufer, wird durch die Rückmeldung TS_ERROR angezeigt. Differenziertere Diagnoseinformation ist dann im Standardkopf verschlüsselt. **tnsx.h** enthält die Definition der Fehlermeldungswerte zu den Aufrufen von ICMX(TNS). Der Standardkopf ist wie folgt definiert:

```
typedef struct {
    short ts_version;     /* Versionsnummer */
    short ts_retcode;     /* Fehlerstyp */
    short ts_errclass;    /* Fehlerklasse */
    short ts_errvalue;    /* Fehlerwert */
} Ts_head;
```

ts_retcode ist der Fehlertyp. Es sind definiert:

```
  0 TS_NOERR    kein Fehler
 -1 TS_TEMPERR  temporärer Fehler
 -2 TS_CALLERR  Aufruffehler
 -3 TS_PERMERR  permanenter Fehler
-99 TS_WARNING  Warnung
```

ts_errclass enthält die Fehlerklasse. Es bedeuten:

```
 0 TS_NOTSPEC  nicht näher spezifizierter Fehler
 1 TS_PARERR   Parameterfehler
 3 TS_ILLVERS  ungültige Version
 4 TS_SYSERR   Systemfehler
 6 TS_INTERR   interner Fehler
99 TS_MESSAGE  Hinweis
```

ts_errvalue enthält den Fehlerwert. Er ist im Zusammenhang mit der Fehlerklasse zu betrachten. Folgende Fehlerwerte sind definiert für **ts_errclass == TS_PARERR**:

```
1 TS_DIRERR   Directory unbekannt
2 TS_NAMERR   der angegebene GLOBALE NAME ist nicht vorhanden
              oder existiert bereits
              (funktionsabhängige Bedeutung)
3 TS_ILLNAM   GLOBALER NAME syntaktisch falsch (Wert eines Namens-
              teils zu lang, falsche Identifikation,
              zu viele Namensteile)

4 TS_STRERR   unzulässiger Wert für "ts11str_def"
5 TS_PROPER   unbekannte Eigenschaft oder keine Eigenschaft
              angegeben oder Eigenschaft existiert bereits
6 TS_SCOPER   unzulässiger Wert für "ts11scope"
9 TS_MEMORY   zur Verfügung gestellter Speicherplatz für die
              Werte von Namensteilen oder Eigenschaften
              unzureichend
12 TS_SIDERR  unbekanntes Kurzkennzeichen
13 TS_ILLUSE  Verwendung eines Kurzkennzeichens nicht erlaubt
15 TS_OBJXST  der angegebene GLOBALE NAME existiert bereits
16 TS_LENERR  unzulässige Länge des Wertes einer Eigenschaft
```

Folgende Fehlerwerte sind definiert für **ts_errclass == TS_INTERR**:

```
 7 TS_TIMOUT   kein DSA vorhanden
 9 TS_MEMORY   vom DSA benötigter Speicherplatz für die
               interne Pufferung oder das DAP (zwischen
               DSA < > DUA) ist im System nicht verfügbar
14 TS_UPDERR   Directory voll (overflow)
20 TS_PROT     Protokollfehler im DAP (zwischen DUA < > DSA)
100 TS_LFILE   fehlerhafte Länge einer Directory-Datei
```

Folgende Fehlerwerte sind definiert für **ts_errclass = = TS_MESSAGE**:

```
 8 TS_TIMLIM     Funktionsausführung abgebrochen wegen
                 Erreichens der Zeitschranke
10 TS_LEAFNO     Anzahl der gefundenen GLOBALEN NAMEN
                 größer als die der erwarteten
11 TS_CHLDNO     Anzahl der gefundenen Namensteile
                 größer als die der erwarteten
```

Im folgenden sind die Aufrufe des TNS mit den möglichen Fehlermeldungen zusammengestellt. Es sind nur die Meldungen aufgeführt, die im TNS explizit erzeugt werden. Meldungen zu SINIX-Systemaufrufen sind nicht näher spezifiziert, sie sind durch "xx errno" gekennzeichnet und sind in **errno.h** [CES] definiert.

ts11_read_leafs

```
 -1 TS_TEMPERR  6 TS_INTERR   9 TS_MEMORY
 -2 TS_CALLERR  1 TS_PARERR   1 TS_DIRERR
                              3 TS_ILLNAM
                              4 TS_STRERR
                              6 TS_SCOPER
                              9 TS_MEMORY
                3 TS_ILLVERS  0 TS_NOTSPEC
 -3 TS_PERMERR  4 TS_SYSERR   xx errno
                6 TS_INTERR   9 TS_MEMORY
                            100 TS_LFILE
-99 TS_WARNING 99 TS_MESSAGE 10 TS_LEAFNO
```

ts12_read_children

```
-1 TS_TEMPERR  6 TS_INTERR   9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR   1 TS_DIRERR
                             2 TS_NAMERR
                             3 TS_ILLNAM
                             9 TS_MEMORY
               3 TS_ILLVERS  0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR  xx errno
               6 TS_INTERR   9 TS_MEMORY
                           100 TS_LFILE
```

ts13_read_properties

```
-1 TS_TEMPERR  6 TS_INTERR    9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR    1 TS_DIRERR
                              2 TS_NAMERR
                              3 TS_ILLNAM
                              9 TS_MEMORY
               3 TS_ILLVERS   0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR   xx errno
               6 TS_INTERR    9 TS_MEMORY
                            100 TS_LFILE
```

ts21_create_non_leaf_entity

```
-1 TS_TEMPERR  6 TS_INTERR   9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR   1 TS_DIRERR
                             2 TS_NAMERR
                             3 TS_ILLNAM
                            15 TS_OBJXST
               3 TS_ILLVERS  0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR  xx errno
               6 TS_INTERR  14 TS_UPDERR
                           100 TS_LFILE
```

ts22_delete_non_leaf_entity

```
-1 TS_TEMPERR  6 TS_INTERR   9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR   1 TS_DIRERR
                             2 TS_NAMERR
                             3 TS_ILLNAM
                            15 TS_OBJXST
               3 TS_ILLVERS  0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR  xx errno
               6 TS_INTERR 100 TS_LFILE
```

ts23_create_leaf_entity

```
-1 TS_TEMPERR  6 TS_INTERR   9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR   1 TS_DIRERR
-2 TS_CALLERR  1 TS_PARERR   2 TS_NAMERR
-2 TS_CALLERR  1 TS_PARERR   3 TS_ILLNAM
                             5 TS_PROPER
                            15 TS_OBJXST
                            16 TS_LENERR
               3 TS_ILLVERS  0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR   xx errno
               6 TS_INTERR  14 TS_UPDERR
                           100 TS_LFILE
```

ts24_delete_leaf_entity

```
-1 TS_TEMPERR  6 TS_INTERR   9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR   1 TS_DIRERR
                             2 TS_NAMERR
                             3 TS_ILLNAM
               3 TS_ILLVERS  0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR   xx errno
               6 TS_INTERR 100 TS_LFILE
```

ts25_add_property

```
-1 TS_TEMPERR  6 TS_INTERR    9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR    2 TS_NAMERR
                              3 TS_ILLNAM
                              1 TS_DIRERR
                              5 TS_PROPER
                             16 TS_LENERR
                             13 TS_ILLUSE
               3 TS_ILLVERS   0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR    xx errno
               6 TS_INTERR   14 TS_UPDERR
                            100 TS_LFILE
```

ts26_delete_property

```
-1 TS_TEMPERR  6 TS_INTERR    9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR    1 TS_DIRERR
                              2 TS_NAMERR
                              3 TS_ILLNAM
                              5 TS_PROPER
                             13 TS_ILLUSE
                             16 TS_LENERR
               3 TS_ILLVERS   0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR    xx errno
               6 TS_INTERR  100 TS_LFILE
```

ts27_change_property

```
-1 TS_TEMPERR  6 TS_INTERR    9 TS_MEMORY
-2 TS_CALLERR  1 TS_PARERR    1 TS_DIRERR
                              2 TS_NAMERR
                              3 TS_ILLNAM
                              5 TS_PROPER
                              5 TS_PROPER
                             13 TS_ILLUSE
                             16 TS_LENERR
               3 TS_ILLVERS   0 TS_NOTSPEC
-3 TS_PERMERR  4 TS_SYSERR    xx errno
               6 TS_INTERR  100 TS_LFILE
```

Konventionen

Bei Verwendung von ICMX(TNS) sind folgende Konventionen einzuhalten:

1) alle Identifier, die mit 'b' beginnen, sind reserviert für die Systemsoftware;

2) alle Identifier, die mit "ts" oder "Ts" beginnen, sind für TNS reserviert;

3) alle Präprozessordefinitionen, die mit oder "TS" beginnen, sind für TNS reserviert.

TNS-Funktionsaufrufe

Die folgenden Seiten beschreiben die TNS-Aufrufe im Detail. Parametern, in denen TNS einen vom Aufrufer bereitgestellten Wert erwartet, sind mit "->" gekennzeichnet; Parameter, in denen der TNS nach dem Aufruf einen Wert liefert, sind mit "<-" gekennzeichnet; Parameter, in denen der Aufrufer einen Wert bereitstellen muß, der dann von TNS modifiziert wird, tragen das Kennzeichen "<>".

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

SIEHE AUCH

```
errno.h - Meldungen zu SINIX-Systemaufrufen
[CCP] - CCP-Manual
[CES] - CES C-Entwicklungsystem
[SIN1] - Betriebssystem SINIX, Buch 1
```

NAME

ts11_read_leafs - Auflisten von GLOBALEN NAMEN mit vorgegebenen Namensteilen und Eigenschaften

SYNOPSIS

#include <tnsx.h>

short ts11_read_leafs (parblk)
Ts_P11 *parblk ;

```
typedef struct {
    Ts_head       *ts_head ;
    short         ts11dir_id ;
    Ts_leaf_name  *ts11partial_name ;
    Ts_str_def    ts11str_def ;
    Ts_filter     *ts11filter ;
    Ts_scope      ts11scope ;
    Ts_pfname     ts11filename ;
    short         ts11time_limit ;
    char          *ts11npval ;
    short         ts11l_np_values ;
    short         ts11exp_no ;
    Ts_leaf_name  *ts11leafs;
    short         ts11real_no ;
} Ts_P11 ;
```

BESCHREIBUNG

Diese Funktion sucht im angegebenen TS-Directory die GLOBALEN NAMEN auf, die die vorgegebenen Namensteile und Eigenschaften besitzen und liefert sie zurück. Die Ergebnisübergabe an den Aufrufer kann im Speicher oder in einer angegebenen Datei erfolgen. Bei Übergabe im Speicher ist das Format durch die entsprechenden Datentypen festgelegt. Bei Übergabe in einer Datei ist das Format weiter unten beschrieben.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_1VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts11dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts11partial_name
Zeiger auf einen Datentyp Ts_leaf_name mit vorgegebenen Namensteilen. Ist ts11partial_name->ts_no_of_el = 0, oder ts11partial_name = NULL, so werden keine Namensteile in die Suche einbezogen, d.h. alle Namensteile passieren den Filter. Sonst werden nur solche GLOBALEN NAMEN gesucht, die mindestens die angegebenen Namensteile besitzen. Besteht der Wert eines Namensteils nur aus einem '*', so werden die GLOBALEN NAMEN gesucht, die diesen Namensteil mit einem beliebigen Wert besitzen.

-> ts11str_def
gibt an, auf welche Art und Weise die Suche nach den vorgegebenen Namensteilen gemäß **ts11partial_name** erfolgen soll. Werte:

TS_RESTRICTED
Nur solche GLOBALEN NAMEN werden gesucht, die genau die angegebenen Namensteile, also genau die angegebene Namensstruktur besitzen.

TS_GENERAL
Nur solche GLOBALEN NAMEN werden gesucht, die mindestens die angegebenen Namensteile besitzen, daneben aber noch weitere, in der Hierarchie tiefer liegende.

-> ts11filter
Zeiger auf einen Datentyp Tslfilter mit einer vorgegebenen Eigenschaft. Ist ts11filter->ts_pr_name = TS_EMPTYPROP, so wird keine Eigenschaft in die Suche einbezogen. Ist ts11filter->ts_pr_length = 0, also eine Eigenschaft der Länge 0 vorgegeben, so werden nur solche GLOBALEN NAMEN gesucht, die mindestens diese Eigenschaft (mit einem beliebigen Wert) haben. Anderenfalls werden solche GLOBALEN NAMEN gesucht, die die angegebene Eigenschaft mit dem angegebenen Wert haben.

-> ts11scope
gibt an, ob die Suche lokal oder global erfolgen soll. Bei TS_LOCAL erfolgt nur eine lokale Suche, bei TS_GLOBAL erfolgt die Suche global.

Hinweis
Globale Suche wird in Version 1 nicht unterstützt.

-> ts11filename
Zeiger auf den Namen einer Datei (Pfadname entsprechend den SINIX-Konventionen), in der vom TNS die gefundenen GLOBALEN NAMEN abgelegt werden sollen (Alternative zur unmittelbaren Rückgabe der gefundenen GLOBALEN NAMEN an den Aufrufer in bereitgestellten Speicherbereichen). Alle Dateiverzeichnisse im Pfadnamen der Datei müssen existieren. Falls die Datei noch nicht vorhanden ist, muß das letzte Dateiverzeichnis im Pfadnamen schreibbar sein. Existiert die Datei, muß sie für schreibenden Zugriff des Besitzers eingerichtet sein, sie wird dann überschrieben. Bei Angabe dieses Parameters braucht in **ts11npval** oder **ts11leafs** kein Speicherbereich bereitgestellt werden.

-> ts11time_limit
gibt die Maximalzeit (in Sekunden) an, die der TNS für die Suche brauchen darf. Nach dieser Zeit bricht er eine nicht vollständig durchgeführte Suche ab.

Hinweis
Funktion wird in Version 1 nicht unterstützt, jede (lokale!) Suche wird vollständig abgeschlossen. Es ist NULL anzugeben.

<- ts11npval
Zeiger auf einen vom Aufrufer bereitzustellenden Bereich, in den vom TNS die Werte der Namensteile der gefundenen GLOBALEN NAMEN eingetragen werden. Angabe NULL, wenn die Übergabe der Ergebnisse gemäß **ts11filename** erfolgt.

-> ts111_np_values
Länge des Bereiches ts11npval. Angabe NULL, wenn die Übergabe der Ergebnisse gemäß ts11filename erfolgt.

-> ts11exp_no
Maximale Anzahl GLOBALER NAMEN, die der TNS in das bereitgestellte Feld **ts11leafs** eintragen kann. Angabe NULL, wenn die Übergabe der Ergebnisse gemäß **ts11filename** erfolgt.

<- ts11leafs
Zeiger auf ein vom Aufrufer bereitzustellendes Feld von Datentypen Ts_leaf_name, in das der TNS die Beschreibungen der gefundenen GLOBALEN NAMEN einträgt. Angabe NULL, wenn die Übergabe der Ergebnisse gemäß **ts11filename** erfolgt.

<- ts11real_no
enthält bei erfolgreichem Abschluß (oder Warnung) die Anzahl der vom TNS gefundenen und zurückgelieferten GLOBALEN NAMEN.

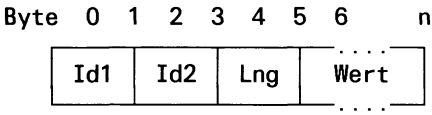
RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

AUSGABEFORMAT

Die Ausgabe der gefundenen GLOBALEN NAMEN in eine Datei erfolgt als Feld von folgenden Einträgen:



Pro Namensteil ist ein Feldelement abgelegt. Die Namensteile eines GLOBALEN NAMENS folgen in absteigender Hierarchie unmittelbar hintereinander. Die Identifikation ID des Namensteils kann aus Id1 und Id2 wie folgt berechnet werden (C-Notation):

ID = Id2 << 5 | Id1

Lng ist die Länge von Wert, Wert sind die Zeichen des Namensteils. Ein Element, in dem die Felder Id1, Id2 und Lng 0 sind, bezeichnet das Ende eines GLOBALEN NAMENS. Danach folgt entweder der erste Namensteil des nächsten GLOBALEN NAMENS oder Dateiende.

Die GLOBALEN NAMEN sind in keiner Weise sortiert, sondern werden, wie im TS-Directory gefunden, abgelegt.

NAME

ts12_read_children - Auflisten der Namensteile der nächstniedrigeren Hierarchiestufe

SYNOPSIS

#include <tnsx.h>

short ts12_read_children (parblk)
Ts_P12 *parblk;

```
typedef struct {
    Ts_head          *ts_head;
    short            ts12dir_id;
    Ts_non_leaf_name *ts12parent;
    Ts_filter        *ts12filter;
    char             *ts12npval;
    short            ts12l_np_values;
    short            ts12exp_no;
    Ts_name_part     *ts12children;
    short            ts12real_no;
} Ts_P12 ;
```

BESCHREIBUNG

Diese Funktion liefert aus dem angegebenen TS-Directory zur NonLeafEntity mit dem angegebenen GLOBALEN NAMEN die daran hängenden Namensteile der nächstniedrigeren Hierarchiestufe zurück. Durch die Vorgabe einer Eigenschaft läßt sich eine Filterung durchführen. Nur solche Namensteile werden zurückgeliefert, die zusammen mit dem vorgegebenen GLOBALEN NAMEN der NonLeafEntity den GLOBALEN NAMEN einer LeafEntity ergeben, dem die angegebene Eigenschaft zugeordnet ist. Mit Hilfe dieser Funktion kann eine Traversierung des Namensbaumes erfolgen.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_1VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts12dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts12parent
Zeiger auf einen Datentyp Ts_non_leaf_name mit dem GLOBALEN NAMEN der NonLeafEntity im Namensbaum, deren nachgeordnete Namenteile aufgelistet werden sollen. TNS meldet einen Fehler, wenn der angegebene GLOBALE NAME nicht vorhanden ist oder keine NonLeafEntity bezeichnet. Bei ts12parent->ts_no_of_el = = 0 werden die unmittelbar an der Wurzel des Namensbaumes hängenden Namensteile aufgelistet.

-> ts12filter
Zeiger auf einen Datentyp Ts_filter mit einer vorgegebenen Eigenschaft. Ist ts12filter->ts_pr_name = TS_EMPTYPROP, so wird keine Eigenschaft in die Auflistung einbezogen. Ist ts12filter->ts_pr_length = 0, also eine Eigenschaft der Länge 0 vorgegeben, so werden nur solche Namensteile geliefert, die zusammen mit **ts12parent** einen GLOBALEN NAMEN mit mindestens dieser Eigenschaft (mit einem beliebigen Wert) beschreiben.

<- s12npval
Zeiger auf einen vom Aufrufer bereitzustellenden Bereich, in den vom TNS die Werte der gefundenen Namensteile eingetragen werden.

-> ts12l_np_values
Länge des Bereiches **ts12npval**.

-> ts12exp_no
Maximale Anzahl der Namensteile, die der TNS in das bereitgestellte Feld **ts12children** eintragen kann.

<- ts12children
Zeiger auf ein vom Aufrufer bereitzustellendes Feld vom Typ Ts_name-part, in das der TNS die Beschreibungen der gefundenen Namensteile einträgt.

<- ts12real_no
enthält bei erfolgreichem Abschluß (oder Warnung) die Anzahl der vom TNS gefundenen und zurückgelieferten Namensteile.

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

NAME

ts13_read_properties - Abfrage der Eigenschaften zu einem GLOBALEN NAMEN

SYNOPSIS

#include <tnsx.h>

short ts13_read_properties (parblk)
Ts_P13 *parblk ;

```
typedef struct {
    Ts_head          *ts_head ;
    short            ts13dir_id ;
    Ts_leaf_name     *ts13leaf_name ;
    long             ts13short_id ;
    char             *ts13prval ;
    short            ts13l_pr_values ;
    Ts_property      *ts13prop ;
    Ts_leaf_name     *ts13dist_name ;
    char             *ts13npval ;
    short            ts13l_np_values ;
} T_P13 ;
```

BESCHREIBUNG

Die Funktion liefert aus dem angegebenen TS-Directory den Wert einer oder mehrerer angegebener Eigenschaften der LeafEntity mit dem angegebenen GLOBALEN NAMEN.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_1VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts13dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts13leaf_name
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_leaf_name mit der Beschreibung des GLOBALEN NAMENS, von dem der Wert einer oder mehrerer Eigenschaften ermittelt werden soll. Der angegebene GLOBALE NAME muß im Namensbaum als LeafEntity vorhanden sein.

<- ts13short_id
Der TNS liefert hier das Kurzkennzeichen für den in **ts13leaf_name** angegebenen GLOBALEN NAMEN zurück. Das Kurzkennzeichen kann dann bei anderen Funktionen alternativ an Stelle dieses GLOBALEN NAMENS angegeben werden.

Hinweis
Kurzkennzeichen werden in Version 1 nicht unterstützt.

<- ts13prval
Zeiger auf einen vom Aufrufer bereitzustellenden Bereich, in den der TNS die Werte der angeforderten Eigenschaften einträgt.

-> ts13l_pr_values
Länge des Bereiches **ts13prval**

<> ts13prop
Zeiger auf ein vom Aufrufer bereitzustellendes Feld von Datentypen Ts_property, in dem vor dem Aufruf die Identifikationen und Typen aller angeforderten Eigenschaften anzugeben sind. Das letzte Element des Feldes wird dadurch gekennzeichnet, daß in ihm als Identifikation der Eigenschaft TS_EMPTYPROP steht. Der TNS trägt in jedem Element in diesem Feld für die angeforderte Eigenschaft die Länge ihres Wertes sowie einen Zeiger auf den in dem Wertefeld **ts13prval** abgelegten Wert ein. Ist eine angeforderte Eigenschaft nicht vorhanden, so trägt der TNS für die Länge des Wertes dieser Eigenschaft den Wert 0 in das Element ein. Der Zeiger ist dann undefiniert.

<- ts13dist_name
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_leaf_name, in den der TNS die Beschreibung des GLOBALEN NAMENS einträgt, mit dem die angesprochene LeafEntity im Namensbaum erstmals erzeugt worden ist. Einer LeafEntity können neben dem ersten GLOBALEN NAMEN, dem **distinguished name** noch weitere Namen, **aliases**, zugeordnet werden, mit denen sie ebenfalls angesprochen werden kann.

Hinweis
Funktion wird in Version 1 nicht unterstützt, Angabe NULL.

<- ts13npval
Zeiger auf einen vom Aufrufer bereitzustellenden Bereich, in den vom TNS die Werte der Namensteile des distinguished name gemäß **ts13dist_name** eingetragen werden.

Hinweis
Funktion wird in Version 1 nicht unterstützt, Angabe NULL.

-> ts13l_np_values
Länge des Bereiches **ts13npval**.

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

NAME

ts21_create_non_leaf_enitity - Einrichten einer NonLeafEntity im Namensbaum

SYNOPSIS

#include <tnsx.h>

short ts21_create_non_leaf_entity (parblk)
Ts_P21 *parblk ;

```
typedef struct {
    Ts_head          *ts_head ;
    short            ts21dir_id ;
    Ts_non_leaf_name *ts21parent ;
} Ts_P21 ;
```

BESCHREIBUNG

Die Funktion erzeugt eine neue (noch nicht vorhandene) NonLeafEntity mit dem angegebenen GLOBALEN NAMEN im angegebenen TS-Directory. Die Funktion dient der internen Strukturierung des TS-Directory.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_2VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts21dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts21parent
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_non_leaf_name mit der Beschreibung des GLOBALEN NAMENS der NonLeafEntity, die im TS-Directory neu angelegt werden soll.

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

NAME

ts22_delete_non_leaf_entity - Löschen einer NonLeafEntity im Namensbaum

SYNOPSIS

#include <tnsx.h>

short ts22_delete_non_leaf_entity (parblk)
Ts_P22 *parblk ;

```
typedef struct {
    Ts_head          *ts_head ;
    short            ts22dir_id ;
    Ts_non_leaf_name *ts22parent ;
} Ts_Ps22 ;
```

BESCHREIBUNG

Die Funktion löscht die NonLeafEntity mit dem angegebenen GLOBALEN NAMEN aus dem angegebenen TS-Directory. Die Funktion läuft nur dann erfolgreich ab, wenn der durch die NonLeafEntity bestimmte Teilbaum leer ist. Anderenfalls muß zuerst (rekursiv) durch entsprechende Aufrufe der Teilbaum gelöscht werden.

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_2VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts22dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts22parent
Zeiger auf einen vom Aufrufer bereitzustellenden Datentypen Ts_non_leaf_name mit der Beschreibung des GLOBALEN NAMENS der NonLeafEntity, die aus dem TS-Directory entfernt werden soll.

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

NAME

ts23_create_leaf_entity - Einrichten einer LeafEntity mit Eigenschaften im Namensbaum

SYNOPSIS

#include <tnsx.h>

short ts23_create_leaf_entity (parblk)
Ts_P23 *parblk ;

```
typedef struct {
     Ts_head          *ts_head ;
     short            ts23dir_id ;
     Ts_non_leaf_name *ts23parent ;
     Ts_name_part     *ts23dist_arc ;
     Ts_property      *ts23prop ;
} Ts_P23 ;
```

BESCHREIBUNG

Die Funktion erzeugt eine neue (noch nicht vorhandene) LeafEntity mit dem angegebenen GLOBALEN NAMEN und den angegebenenen Eigenschaften im angegebenen TS-Directory. Die entsprechende NonLeafEntity der nächsthöheren Hierarchiestufe muß bereits vorhanden sein. Mindestens eine Eigenschaft ist anzugeben.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_2VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts23dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts23parent
Zeiger auf einen vom Aufrufer bereitzustellenden Datentypen Ts_non_leaf_name mit der Beschreibung des GLOBALEN NAMENS der NonLeafEntity, an die die neue LeafEntity im TS-Directory angehängt werden soll. Die NonLeafEntity muß bereits existieren.

-> ts23dist_arc
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_name_part mit der Beschreibung des neuen Namensteils, der an die NonLeafEntity **ts23parent** angehängt werden soll. Er darf noch nicht vorhanden sein und muß von niedrigerer Hierarchie sein als der letzte Namensteil in **ts23parent**.

-> ts23prop
Zeiger auf ein vom Aufrufer bereitzustellendes Feld von Datentypen Ts_property mit der Beschreibung der Eigenschaften, die der neuen LeafEntity zugeordnet werden sollen. Das letzte Element des Feldes muß dadurch gekennzeichnet sein, daß in ihm die Identifikation der Eigenschaft den Wert TS_EMPTYPROP besitzt.

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

NAME

ts24_delete_leaf_entity - Löschen einer LeafEntity im Namensbaum

SYNOPSIS

#include <tnsx.h>

short ts24_delete_leaf_entity (parblk)
Ts_P24 *parblk ;

```
typedef struct {
    Ts_head         *ts_head ;
    short           ts24dir_id ;
    Ts_leaf_name    *ts24dist_name ;
    } Ts_P24 ;
```

BESCHREIBUNG

Die Funktion löscht die LeafEntity mit dem angegebenen GLOBALEN NAMEN samt den zugehörigen Eigenschaften aus dem angegebenen TS-Directory.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_2VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts24dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts24dist_name
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_leaf_name mit der Beschreibung des GLOBALEN NAMENS der LeafEntity, die aus dem TS-Directory entfernt werden soll.

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

NAME

ts25_add_property - Einrichten einer Eigenschaft einer LeafEntity

SYNOPSIS

#include <tnsx.h>

short ts25_add_property (parblk)
Ts_P25 *parblk ;

```
typedef struct {
    Ts_head      *ts_head ;
    short        ts25dir_id ;
    Ts_leaf_name *ts25dist_name ;
    long         ts25short_id ;
    Ts_property  *ts25prty ;
} Ts_P25 ;
```

BESCHREIBUNG

Die Funktion ordnet im angegebenen TS-Directory der LeafEntity mit dem angegebenen GLOBALEN NAMEN die angegebene neue Eigenschaft zu. Die LeafEntity muß vorhanden sein, die Eigenschaft darf noch nicht vorhanden sein.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_2VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts25dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts25dist_name
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_leaf_name mit der Beschreibung des GLOBALEN NAMENS der LeafEntity, der eine neue Eigenschaft zugeordnet werden soll.

-> ts25short_id
ist ein Kurzkennzeichen. Dieses vom TNS bei der Funktion **ts23_read_properties** zurückgelieferte Kennzeichen kann an Stelle des GLOBALEN NAMENS in **ts25dist_name** angegeben werden.

Hinweis
Kurzkennzeichen werden in Version 1 nicht unterstützt, Angabe NULL.

-> ts25prty
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_property mit der Beschreibung der Eigenschaft, die der angegebenen LeafEntity neu zugeordnet werden soll.

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

NAME

ts26_delete_property - Löschen einer Eigenschaft einer LeafEntity

SYNOPSIS

#include <tnsx.h>

short ts26_delete_property (parblk)
Ts_P26 *parblk ;

```
typedef struct {
    Ts_head         *ts_head ;
    short           ts26dir_id ;
    Ts_leaf_name    *ts26dist_name ;
    long            ts26short_id ;
    short           ts26prid ;
} Ts_P26 ;
```

BESCHREIBUNG

Die Funktion löscht die angegebene Eigenschaft der LeafEntity mit dem angegebenen GLOBALEN NAMEN im angegebenen TS-Directory. Die Eigenschaft muß vorhanden sein.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_2VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts26dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts26dist_name
Zeiger auf einen vom Aufrufer bereitzustellenden Datentypen Ts_leaf_name mit der Beschreibung des GLOBALEN NAMENS der LeafEntity, von der eine Eigenschaft gelöscht werden soll.

-> ts26short_id
Kurzkennzeichen. Dieses vom TNS bei der Funktion **ts23_read_properties** zurückgelieferte Kennzeichen kann an Stelle des GLOBALEN NAMENS in **ts26dist_name** angegeben werden.

Hinweis
Kurzkennzeichen werden in Version 1 nicht unterstützt, Angabe NULL.

-> ts26prid
Identifikation der Eigenschaft, die gelöscht werden soll.
Werte: siehe Include-Datei tnsx.h

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

NAME

ts27_change_property - Ändern einer existierenden Eigenschaft einer LeafEntity

SYNOPSIS

#include <tnsx.h>

short ts27_change_property (parblk)
Ts_P27 *parblk ;

```
typedef struct {
    Ts_head         *ts_head ;
    short           ts27dir_id ;
    Ts_leaf_name    *ts27dist_name ;
    long            ts27short_id ;
    Ts_property     *ts27prty ;
} Ts_P27 ;
```

BESCHREIBUNG

Die Funktion ändert den Wert der angegebenen Eigenschaft der LeafEntity mit dem angegebenen GLOBALEN NAMEN im angegebenen TS-Directory. Die Eigenschaft muß existieren.

Beim Aufruf ist ein Zeiger auf den dargestellten, ausgefüllten Parameterblock zu übergeben.

Parameter im Parameterblock:

-> ts_head
Zeiger auf den Standardkopf. Dieser muß vor dem Aufruf in ts_version TS_2VER01 enthalten. Nach Rückkehr mit TS_ERROR stehen darin Diagnoseinformationen.

-> ts27dir_id
Identifikation des TS-Directory. Mögliche Werte: 0 - 9

Hinweis
CMX verwendet intern nur das TS-Directory TS_CMX_DIR_ID.

-> ts27dist_name
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_leaf_name mit der Beschreibung des GLOBALEN NAMEN der LeafEntity, deren Eigenschaft geändert werden soll.

-> ts27short_id
Kurzkennzeichen. Dieses vom TNS bei der Funktion **ts13_read_properties** zurückgelieferte Kennzeichen kann an Stelle des Objektnamens in **ts27dist_name** angegeben werden.

Hinweis
Kurzkennzeichen werden in Version 1 nicht unterstützt, Angabe NULL.

-> ts27prty
Zeiger auf einen vom Aufrufer bereitzustellenden Datentyp Ts_property mit der Beschreibung der Eigenschaft. Der angegebene Wert der Eigenschaft ersetzt den vorhandenen Wert, er kann kürzer oder länger als der vorhandene Wert sein.

RÜCKMELDUNGEN

TS_OK
Funktion erfolgreich abgeschlossen.

TS_ERROR
Funktion mit Fehler abgeschlossen. Der Standardkopf enthält detaillierte Informationen über den aufgetretenen Fehler.

6.3 ICMX(NEA) - Programmschnittstelle zum Migrationsservice NEABX

Die folgenden Seiten beschreiben die Programmschnittstelle ICMX(NEA) zum Migrationsservice in SINIX. Sie enthalten

- einen Überblick über die Funktionen der Schnittstelle ICMX(NEA) mit Details zu den Kommunikationsphasen und Hinweisen zur korrekten Verwendung der Funktionen (Zustandsautomaten) sowie
- die präzise Beschreibung der Funktionsaufrufe mit allen Parametern.

Die Schnittstelle ICMX(NEA) ist ein Migrations-Service, der die Kommunikation zwischen CMX-Anwendungen in SINIX und bestehenden Anwendungen im TRANSDATA-Netz ermöglicht, ohne daß diese Anwendungen Änderungen an ihren Kommunikations-Schnittstellen vornehmen müssen. Daher bietet ICMX(NEA) auch Funktionen, die über die Funktionalität eines ISO-Transportsystems nach ISO-Norm 8072 hinausgehen, um den Anforderungen bestehender Anwendungen im TRANSDATA-Netz zu genügen.

Die erweiterte Funktionalität hat andererseits auch unterschiedliche Funktionsaufrufe und Optionen im Vergleich zu den Funktionsaufrufen von ICMX(L) zur Folge. Wie schon im Abschnitt 1.2.3 beschrieben, bietet ICMX(NEA) insbesondere in der Verbindungsaufbauphase Funktionen, die im NEABX-Protokoll angeboten werden, aber im ISO-Transportsystem nach ISO-Norm 8072 nicht verfügbar sind. Einen Überblick über die wesentlichen Unterschiede zwischen einander entsprechenden Aufrufen von ICMX(L) (mit t_...) und ICMX(NEA) (mit x_...) finden Sie hier in der Kurzbeschreibung der Funktionsaufrufe bei den jeweiligen Kommunikationsphasen.

Transport Service

NEABX bietet in der vorliegenden Version CMX V2.1 mit ICMX(NEA) eine Programmschnittstelle zum verbindungsorientierten Transport Service (TS). Der TS erlaubt es zwei Anwendungen (TS-Anwendungen), Nachrichten über eine Transportverbindung (TV) auszutauschen. Die verbindungsorientierte Kommunikation bietet Nachrichtenaustausch in zuverlässiger Weise unter Beibehaltung der Nachrichtenreihenfolge. Ferner ermöglicht der verbindungsorientierte TS über Verbindungsidentifikationen den Verzicht auf Adreßübertragung und -verarbeitung in der Datenphase. Zur korrekten Funktion der Kommunikation müssen gewisse Regeln beachtet werden, die im folgenden beschrieben werden.

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

Gewisse Parameter, die den Transport der Nachrichten auf der TV beeinflussen, müssen von den TS-Anwendungen beim Verbindungsaufbau verhandelt werden.

Für die TV ist die Transportreferenz wesentlich. Das ist eine Zahl, die eine TV beim Auftragsaustausch zwischen NEABX und TS-Anwendung eindeutig identifiziert. Intern ist jeder TV eine exklusiv eröffnete SINIX-Gerätedatei zugeordnet, die für die TS-Anwendung aber nicht sichtbar wird. Die exklusive Eröffnung vereinfacht in NEABX die Aufräumungsmaßnahmen bei einer vorzeitigen Beendigung der TS-Anwendung.

Fehlerbehandlung

Alle Funktionsaufrufe enden mit einer Rückmeldung. Diese zeigt zum Beispiel mit X_OK den erfolgreichen Abschluß an oder informiert mit X_ERROR pauschal über einen aufgetretenen Fehler. Detailliertere Diagnoseinformationen erhalten Sie anschließend durch die Fehlerabfragefunktion x_error(), die Sie sofort nach dem Auftreten eines Fehlers aufrufen müssen.

Alle Fehlermeldungen, die von NEABX als Nichteinhaltung der Kommunikationsregeln durch die TS-Anwendung erkannt werden, haben einen spezifischen Fehlercode und sind in neabx.h, cmx.h oder tnsx.h definiert. Andere Fehler resultieren aus Mißerfolgen beim Aufruf von Funktionen der Betriebssystemumgebung in CMX, sie sind in errno.h beschrieben. Die verwendeten Transportsysteme erzeugen keine Fehlermeldungen, eventuelle Fehlerfälle führen zum Verbindungsabbau mit einem entsprechenden Abbaugrund.

TS-Anwendungen, Transportverbindungen und SINIX-Prozesse

Eine TS-Anwendung ist ein System von Programmen, das den TS, also die Dienste von NEABX, "anwendet". Die Abbildung der TS-Anwendung auf das SINIX-Prozeßkonzept bleibt dem Implementierer überlassen. Eine TS-Anwendung kann sich in einem oder mehreren (nicht notwendig verwandten) Prozessen organisieren. Die Prozesse können prinzipiell unabhängig voneinander TVen zu fernen TS-Anwendungen unterhalten. Die Prozesse einer TS-Anwendung können ihre TVen untereinander austauschen. Die Transportreferenz einer TV ist jedoch zu jedem Zeitpunkt genau einem Prozeß zugeordnet, sie kann deshalb nicht an Kinderprozesse vererbt werden. Es gibt in NEABX einen eigenen lokalen Dienst REDIRECT zum Umlenken einer TV an einen anderen Prozeß.

Ein Prozeß 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. NEABX unterstützt dies durch den asynchronen Verarbeitungsmodus.

Synchronität und Asynchronität, TS-Ereignisse

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

Die Funktionen von NEABX 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 NEABX angenommen und der TS-Anwendung bei nächster Gelegenheit auf Anfrage zugestellt.

NEABX 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 der aufrufende Prozeß suspendiert, bis ein TS-Ereignis eintrifft. Dieses weckt den Prozeß, damit er das TS-Ergebnis sofort verarbeiten kann. Damit ein synchron wartender Prozeß aber nicht endlos wartet, kann er auch durch Signale wie SIGALRM geweckt werden. In diesem Falle setzt ihn NEABX mit dem TS-Ergebnis X_NOEVENT fort. Der synchrone Mechanismus ist nützlich für TS-Anwendungen, die zwischen zwei TS-Ergebnissen nichts tun können.

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

Die entsprechende Funktion in NEABX ist

x_event
: Bei Übergabe des Parameterwertes X_WAIT wird der Prozeß bis zum Eintreten eines TS-Ereignisses suspendiert. Beim Eintreffen eines Signals wird der suspendierte Prozeß geweckt und kehrt mit X_NOEVENT oder X_ERROR zurück.
Tritt ein TS-Ereignis ein oder liegt ein Fehler vor, so liefert die Funktion sofort den Code des TS-Ereignisses bzw. X_ERROR als Rückmeldung.
Bei Übergabe des Parameterwertes X_CHECK kehrt die Funktion immer sofort zurück und liefert X_NOEVENT bzw. den Code des vorliegenden TS-Ereignisses oder X_ERROR als Rückmeldung.

Abweichungen gegenüber t_event:
: Für x_event sind drei weitere asynchrone Ereignisse definiert: X_REPCRQ, X_REPCIN und X_REPCCF.

Folgende dreizehn asynchrone TS-Ereignisse sind in NEABX definiert:

X_NOEVENT
im asynchronen Fall: kein TS-Ereignis vorhanden;
im synchronen Fall: Abbruch der Funktion durch ein Signal.

X_REPCRQ
NEABX fordert zur Wiederholung der Verbindungsanforderung auf.

X_CONIN
Anzeige einer von einer rufenden TS-Anwendung eintreffenden Verbindungsanforderung.

X_REPCIN
NEABX zeigt die Fortsetzung einer Verbindungsanforderung an, die noch nicht vollständig übergeben wurde.

X_CONCF
Anzeige einer von einer gerufenen TS-Anwendung eintreffenden Verbindungsbestätigung.

X_REPCCF
NEABX zeigt die Fortsetzung einer Verbindungsbestätigung an, die noch nicht vollständig übergeben wurde.

X_DISIN
Anzeige eines von einer fernen TS-Anwendung eintreffenden oder von NEABX veranlaßten Verbindungsabbaus.

X_REDIN
Anzeige einer von einem Prozeß derselben TS-Anwendung eintreffende Verbindungsumlenkung. (Dieses TS-Ergebnis ist lokal, es ist eine Erweiterung des TS's zur Flexibilisierung der Implementierung von TS-Anwendungen.)

X_DATAIN
Von einer TS-Anwendung gesendete gewöhnliche Daten sind eingetroffen.

X_XDATIN
Von einer TS-Anwendung gesendete Vorrangdaten sind eingetroffen.

X_DATAGO
Eine durch die Flußregelung gesetzte Sendesperre für gewöhnliche Daten wird aufgehoben.

X_XDATGO
Eine durch die Flußregelung gesetzte Sendesperre für Vorrangdaten wird aufgehoben.

X_ERROR
Fataler Fehler; nähere Information liefert die Abfragefunktion x_error().

Bei jedem TS-Ereignis (außer X_NOEVENT und X_ERROR) wird der TS-Anwendung auch die Transportreferenz mitgeteilt, damit sie spezifisch für diese TV auf das TS-Ereignis reagieren kann.

TS-Ereignisse werden in der Regel in der Reihenfolge ihres Auftretens zugestellt. Allerdings kann das TS-Ereignis X_XDATIN das TS-Ereignis X_DATAIN überholen und X_DISIN kann X_DATAIN und X_XDATIN überholen. Im letzteren Falle werden die überholten TS-Ereignisse auf dieser TV vernichtet.

Kommunikationsphasen

Der TS in NEABX unterscheidet vier Phasen: Aktivierung/Deaktivierung, Anmeldung/Abmeldung, Verbindungsaufbau und -abbau, Datenaustausch. Zustandsautomaten, die die zulässige Reihenfolge für Funktionsaufrufe in diesen Phasen definieren, zeigen die Bilder 6-7 bis 6-12.

Aktivierung/Deaktivierung

In dieser Phase wird ein Prozeß, in dem ein TS-Anwendungsprogramm abgearbeitet wird, erzeugt oder aufgelöst.

Anmeldung/Abmeldung

In dieser Phase wird die Prozeßumgebung für die Kommunikation über NEABX hergestellt, wenn sich die TS-Anwendung bei NEABX zur Kommunikation anmeldet. Dabei wird ein Dienstzugriffspunkt (Transport Service Access Point, TSAP) eingerichtet, an dem der TS zur Verfügung steht. Dem TSAP ist ein LOKALER NAME zugeordnet, unter dem die TS-Anwendung in diesem Endsystem erreichbar ist. Außerdem wird eine SINIX-Gerätedatei für den Austausch von Aufträgen zwischen den NEABX-Bibliotheksfunktionen und dem Betriebssystem eröffnet. Bei der Abmeldung werden noch bestehende TVen und der TSAP abgebaut, die Prozeßumgebung aufgelöst und belegte Betriebsmittel für zukünftige Verwendung freigegeben.

Ein Prozeß kann gleichzeitig mehrere TSAPs verwalten und an diesen mehrere Verbindungsendpunkte (Transport Connection Endpoint, TCEP) unterhalten. Auch können mehrere Prozesse denselben TSAP verwenden und an diesem aktiv TVen aufbauen oder passiv auf Verbindungsanzeigen warten, ohne zu interferieren. Allerdings ist jeder TCEP genau einem Prozeß 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.

x_attach
: meldet für den laufenden Prozeß eine TS-Anwendung bei NEABX an. Der Prozeß kann bei der Anmeldung sein künftiges Verhalten in dieser TS-Anwendung spezifizieren. Mit der ersten Anmeldung beginnt NEABX für diese TS-Anwendung Verbindungsaufbauanzeigen entgegenzunehmen.

x_detach
: meldet für den laufenden Prozeß eine TS-Anwendung bei NEABX ab. Zugleich baut NEABX die noch bestehenden TVen des Prozesses in dieser TS-Anwendung ab. Sofern kein weiterer Prozeß dieser TS-Anwendung angemeldet ist, ist die TS-Anwendung danach bei NEABX nicht mehr bekannt.

Verbindungsaufbau und -abbau

In dieser Phase bauen zwei TS-Anwendungen eine TV zueinander auf oder ab. Eine der beiden 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), nimmt sie an oder weist sie ab. Während des Verbindungsaufbaus verhandeln die TS-Anwendungen gewisse Merkmale der TV für die kommende Datenphase und können Benutzerdaten austauschen.

Jede der TS-Anwendungen kann jederzeit den Abbau der TV anfordern. Dies wird von den TS-Anwendungen nicht verhandelt, sondern von NEABX sofort vollzogen. Der anderen TS-Anwendung oder - wenn NEABX die TV abbaut - beiden TS-Anwendungen wird eine Verbindungsabbauanzeige zugestellt, die weder beantwortet noch abgewehrt werden kann. Auch NEABX kann jederzeit die TV abbauen, alle Fehlerfälle in den Transportsystemen werden auf diese Weise angezeigt. NEABX garantiert nicht, daß Daten, die zum Zeitpunkt der Anforderung des Abbaus noch unterwegs sind, noch zugestellt werden.

Die entsprechenden Funktionen sind:

x_conrq
: fordert den Verbindungsaufbau zur gerufenen TS-Anwendung mit der angegebenen TRANSPORTADRESSE an. 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 eine Antwort der gerufenen TS-Anwendung warten (siehe oben).

Abweichungen gegenüber t_conrq:

1. Bei x_conrq ist für die Option die Angabe NULL verboten.
2. Bei der Benutzerdatenlänge sind außer der Nettodatenlänge noch 8 Byte Platzreservierung zu berücksichtigen.
3. Es gibt zwei weitere Rückmeldungen: X_DATASTOP und X_REPEAT.

x_conin

übernimmt von NEABX die mit X_CONIN angezeigte Verbindungsanforderung der rufenden TS-Anwendung mit deren TRANSPORT-ADRESSE. Der Bezug zum TSAP wird für die gerufene TS-Anwendung durch Lieferung des bei der Anmeldung angegebenen LOKALEN NAMENS hergestellt.

Abweichungen gegenüber t_conin:

1. Bei x_conin ist für die Option die Angabe NULL verboten.
2. Der bereitestellte Benutzerdatenbereich muß mindestens X_MSG_SIZE Byte groß sein.
3. Es gibt eine weitere Rückmeldung: X_REPEAT.

x_conrs

beantwortet (akzeptiert) die Verbindungsanforderung, nachdem sie mit X_CONIN angezeigt und von NEABX übernommen wurde.

Abweichungen gegenüber t_conrs:

1. Bei x_conrs ist für die Option die Angabe NULL verboten.
2. Bei der Benutzerdatenlänge sind außer der Nettodatenlänge noch 8 Byte Platzreservierung zu berücksichtigen.
3. Es gibt eine weitere Rückmeldung: X_DATASTOP.

x_concf

übernimmt von NEABX die mit X_CONCF angezeigte Antwort der gerufenen TS-Anwendung. Damit ist der Verbindungsaufbau vollzogen.

Abweichungen gegenüber t_concf:

1. Bei x_concf ist für die Option die Angabe NULL verboten.
2. Der bereitgestellte Benutzerdatenbereich muß mindestens X_MSG_SIZE Byte groß sein.
3. Es gibt eine weitere Rückmeldung: X_REPEAT.

x_disrq
fordert den Abbau einer Verbindung an. Die Funktion kann jederzeit von jeder TS-Anwendungen aufgerufen werden. Eine durch NEABX angezeigte und übernommene Anforderung zum Verbindungsaufbau kann, falls sie nicht akzeptiert wird, mit dieser Funktion abgelehnt werden.

x_disin
übernimmt von NEABX die mit X_DISIN angezeigte Verbindungsabbauanzeige. Durch diesen Funktionsaufruf erhält die TS-Anwendung auch den Abbaugrund mitgeteilt.

Datenaustausch, Flußregelung, Verbindungsumlenkung

Nachdem eine TV aufgebaut ist, kann über sie der Transfer von gewöhnlichen und (optional) Vorrangdaten erfolgen. Der Transfer findet nachrichtenorientiert 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 Byte beschränkt, sie werden mit Vorrang zum Strom der gewöhnlichen Daten übertragen und in eigenen Warteschlangen geführt. NEABX garantiert nur, daß Vorrangdaten nie später als danach versendete gewöhnliche Daten bei der empfangenden TS-Anwendung eintreffen. Pro Aufruf kann an NEABX höchstens eine komplette ETSDU übertragen werden.

Die Übergabe einer Nachricht an NEABX muß in Portionen der Länge einer Transport Interface Data Unit (TIDU) - Dateneinheit - erfolgen, wenn die Nachricht länger als eine Dateneinheit ist. Die Nachricht muß dann mit mehreren Sendeaufrufen übertragen werden. Ein Parameter beim Sendeaufruf zeigt an, ob eine weitere Dateneinheit für diese Nachricht erfolgt (X_MORE) oder nicht (X_END). Daraus ist nicht ableitbar, wie die Dateneinheit für die Übertragung oder Zustellung zur empfangenden TS-Anwendung verpackt wird. NEABX garantiert nur, daß die sequentielle Zusammenfügung der Dateneinheiten auf Empfangsseite wieder die Nachricht der Sendeseite liefert. Die Dateneinheitenlänge kann für beide TS-Anwendungen verschieden sein und hängt von der TV ab. Sie muß bei NEABX für jede TV abgefragt werden. NEABX garantiert nicht, daß bei der empfangenden TS-Anwendung jede außer der letzten Dateneinheit einer Nachricht maximal gefüllt zugestellt wird.

Die empfangende TS-Anwendung erhält die Ankunft einer Dateneinheit und von Vorrangdaten über das TS-Ereignis X_DATAIN bzw. X_XDATIN angezeigt. Sie holt dann die Dateneinheit bzw. Vorrangdaten mit einem entsprechenden Funktionsaufruf vollständig ab.

Die Übertragung von Dateneinheiten (Vorrangdaten) unterliegt Flußregelungsmechanismen, die von NEABX und den TS-Anwendungen geregelt werden können. Die Rückmeldung X_DATASTOP (X_XDATSTOP) beim Senden sagt der sendenden TS-Anwendung, daß die Dateneinheit (Vorrangdaten) erfolgreich abgesetzt, der Fluß der Dateneinheiten (Vorrangdaten) aber gesperrt wurde. Es können keine Dateneinheiten (Vorrangdaten) mehr gesendet werden, bis der Fluß wieder freigegeben wird. Dies wird durch das TS-Ereignis X_DATAGO (X_XDATGO) angezeigt.

Die empfangende TS-Anwendung sperrt und entsperrt den Fluß der Dateneinheiten (Vorrangdaten) durch Funktionsaufrufe an NEABX, die sich für die sendende TS-Anwendung wie oben beschrieben, auswirken.

Die Verbindungsumlenkung ist ein lokaler Dienst in NEABX, der die Organisation der TS-Anwendung in Prozesse vereinfacht. Ein Prozeß, der eine fertig aufgebaute TV hält, kann diese (allerdings abhängig vom Zustand, siehe Bilder 6-9 bis 6-12) an einen anderen Prozeß derselben TS-Anwendung umlenken. Der TSAP und der TCEP bleiben dabei unverändert. Der abgegebene Prozeß verliert dabei die Transportreferenz dieser TV, womit sie für ihn nicht mehr verfügbar ist.

Die folgenden Funktionen realisieren Datenaustausch, Flußregelung und Verbindungsumlenkung:

x_info
: liefert für eine etablierte TV die maximale Länge einer Dateneinheit (TIDU-Länge).

x_datarq
: fordert die Übertragung einer (evtl. teilgefüllten) Dateneinheit an. Die Rückmeldung X_DATASTOP besagt, daß der Datenfluß gesperrt ist. Weitere Sendeanforderungen werden solange mit Fehler abgewiesen, bis der Datenfluß wieder freigegeben wird.

Abweichungen gegenüber t_datarq:

1. Falls NEABX für die Datenphase vereinbart wurde, ist bei der ersten Dateneinheit einer Nachricht die Angabe einer Option möglich.
2. Die anzugebende Datenlänge umfaßt Nettodaten und 5 Byte Platzreservierung.

x_datain

übernimmt die Daten einer Dateneinheit von NEABX, nachdem sie mit X_DATAIN angezeigt wurden.

Abweichungen gegenüber t_datain:

1. Falls NEABX für die Datenphase vereinbart wurde, ist bei der ersten Dateneinheit die Auswertung einer Option möglich.
2. Als Datenlänge ist mindestens die Länge einer Dateneinheit anzugeben (siehe x_info()). Eine Dateneinheit wird somit immer vollständig übernommen.
3. Es gibt eine weitere Rückmeldung: X_DATASTOP.

x_xdatrq

fordert die Übertragung von Vorrangdaten an. Die Rückmeldung X_XDATSTOP besagt, daß der Fluß der Vorrangdaten gesperrt ist. Weitere Sendeanforderungen werden solange mit Fehler abgewiesen, bis der Fluß wieder freigegeben wird.

Abweichungen gegenüber t_xdatrq:

1. Falls NEABX für die Datenphase vereinbart wurde, ist die Angabe einer Option möglich.
2. Die anzugebende Datenlänge umfaßt Nettodaten und 5 Byte Platzreservierung.
3. Es gibt eine weitere Rückmeldung: X_XDATSTOP.

x_xdatin

übernimmt die Vorrangdaten von NEABX, nachdem sie mit X_XDATIN angezeigt wurden.

Abweichungen gegenüber t_xdatin:

1. Falls NEABX für die Datenphase vereinbart wurde, ist die Angabe einer Option möglich.
2. Der bereitgestellte Vorrangdatenbereich muß mindestens X_EXP_SIZE Byte groß sein. Vorrangdaten werden also immer vollständig übernommen.
3. Es gibt eine weitere Rückmeldung: X_DATASTOP.

x_redrq
lenkt die TV an einen Prozeß derselben TS-Anwendung um. Dem abgegebenen Prozeß ist sie anschließend nicht mehr verfügbar.

Abweichungen gegenüber t_redrq:

1. Bei x_redrq ist für die Option die Angabe NULL verboten.
2. Die angegebene Datenlänge enthält Nettodaten und 2 Byte Platzreservierung.
3. Es gibt eine weitere Rückmeldung: X_IMPOSSIBLE.

x_redin
übernimmt von NEABX die mit X_REDIN angezeigte Verbindungsumlenkung. Der empfangende Prozeß muß sie annehmen, kann sie aber sofort weitergeben (auch zurückgeben) oder abbauen.

Abweichungen gegenüber t_redin:

1. Bei x_redin ist für die Option die Angabe NULL verboten.
2. Der bereitgestellte Benutzerdatenbereich muß mindestens X_RED_SIZE Byte groß sein.

Zustandsautomaten

Die Abläufe zur Nutzung der Schnittstelle ICMX(NEA) können mit Zustandsautomaten dargestellt werden. Das sind Diagramme, die die definierten Zustände einer TS-Anwendung und die legalen Zustandsübergänge enthalten.

Die Abläufe können in vier Phasen aufgeteilt werden. Diese bilden eine hierarchische Struktur,

Phase A: Aktivierung/Deaktivierung
Phase B: Anmeldung/Abmeldung
Phase C: Verbindungsaufbau/-abbau
Phase D: Datenaustausch

wobei die Phase A der höchsten und die Phase D der niedrigsten Hierarchiestufe entspricht.

Jede Phase wird durch einen oder mehrere Automaten dargestellt. Die doppelt gezeichneten Rechtecke eines Automaten stellen den Zustand dar, in dem die Automaten der nächsten Phase bzw. niedrigeren Hierarchiestufe aktiviert werden. Der Übergang aus einem Zustand, der durch ein doppelt gepunktetes Rechteck dargestellt ist (siehe Bild 6-9), in einen anderen Zustand bewirkt, daß die zugeordneten Automaten der niedrigeren Hierarchiestufe inaktiviert werden.

In Phase A muß ein Prozeß vorhanden sein, der die Schnittstelle ICMX(NEA) bedienen kann. In dieser Phase wird der Prozeß kreiert und zerstört.

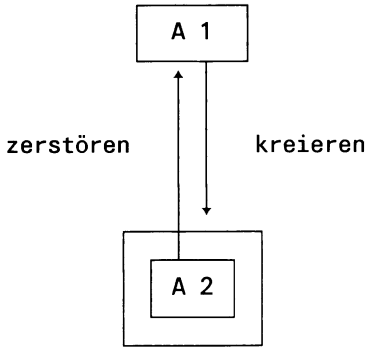


Bild 6-7 Aktivierung/Deaktivierung

In Phase B werden TS-Anwendungen an- oder abgemeldet. Die Anzahl der Automaten in dieser Phase ist gleich der Anzahl der TS-Anwendungen, die der Prozeß durchführt. In B2 hat der Prozeß einen (weiteren) Dienstzugriffspunkt (TSAP) eingerichtet.

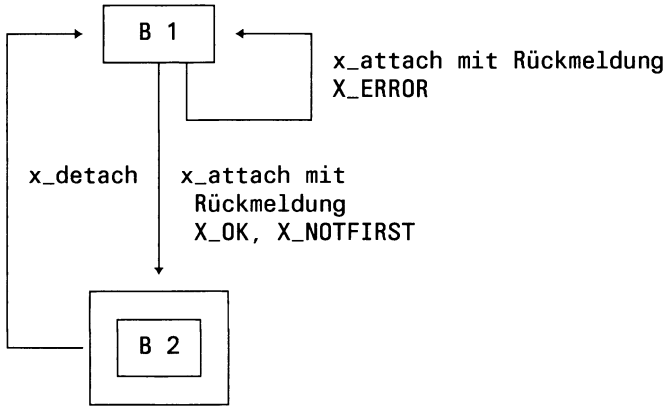


Bild 6-8 An-/Abmeldung von TS-Anwendungen

In Phase C werden TVen auf- und abgebaut und umgelenkt. Die Anzahl der Automaten ist gleich der Anzahl der TVen, die der Prozeß bedient. In C2 hat die TS-Anwendung eine TV aktiv angefordert und wartet auf die Antwort der gerufenen TS-Anwendung, in C3 hat die TS-Anwendung passiv eine TV-Anforderung entgegengenommen, in C4 ist die Verbindung etabliert. In den mit "+" gekennzeichneten Zuständen (nach X_REPEAT) wartet die TS-Anwendung auf die Wiederholungsanforderung, in den mit "*" gekennzeichneten Zustände kann die Wiederholung erfolgen.

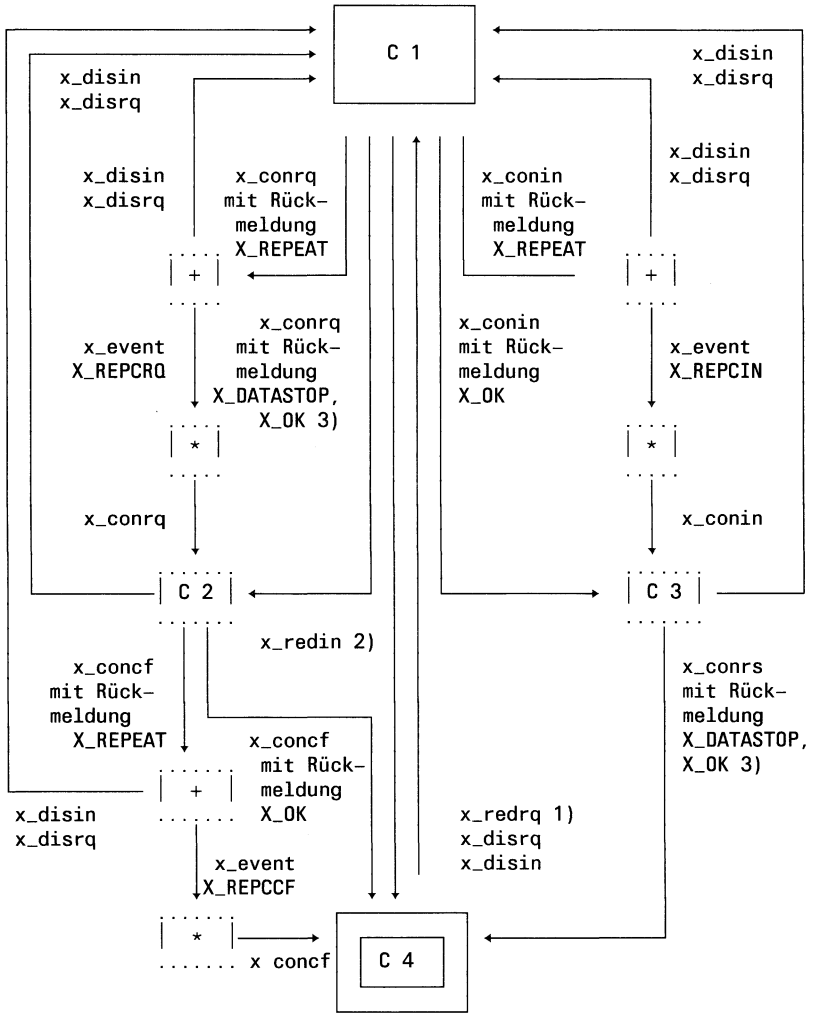


Bild 6-9 Auf- und Abbau und Umlenkung von TVen

1) x_redrq ist nur dann erlaubt, wenn der entsprechende Datensendeautomat in D1 und die Datenempfangsautomaten in D1 und D4 sind.
2) x_redin bewirkt, daß die Datenautomaten in die Zustände übergehen, die den Zuständen der Datenautomaten des Prozesses entsprechen, der das x_redrq initiiert hat.
3) x_conrq und x_conrs mit Rückgabewert X_DATASTOP wirkt in der Verbindungsphase wie X_OK, der zugehörige Datensendeautomat geht zugleich in den Zustand D2 über.

Phase D ist die Datenphase. Sie wird durch 2 parallele Automaten (Datensendeautomat, Datenempfangsautomat) dargestellt. Es gibt also n Automaten, wobei n = 2 * Anzahl der TVen ist. In D2 ist der Datenfluß für Normaldaten gestoppt, in D3 auch der Fluß der Vorrangdaten.

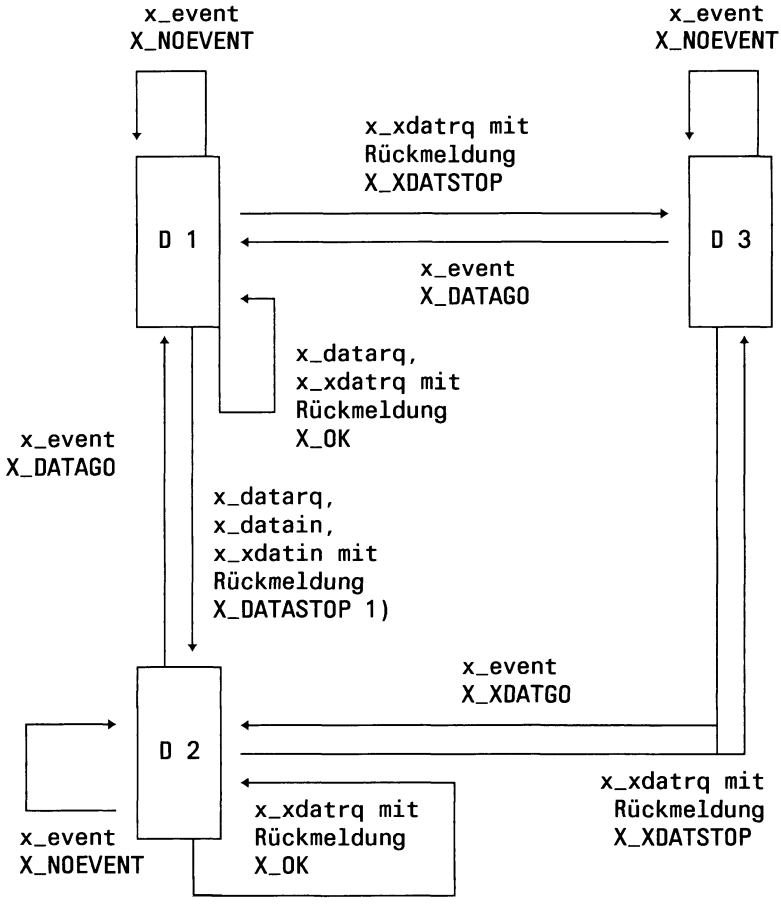


Bild 6-10 Datensendeautomat für Normal- und Vorrangdaten

1) Bei Rückmeldung X_DATASTOP nach x_conrq oder x_conrs geht der zugehörige Datensendeautomat in den Zustand D2 über.

Der Datenempfangsautomat wird der Übersichtlichkeit halber in zwei getrennten Automaten (für Normaldaten bzw. für Vorrangdaten) dargestellt. Es ist zu beachten, daß x_event(), x_xdatstop() bzw. x_datago() auf beide Automaten wirkt. In D2 können zum Empfangen bereitstehende Daten bzw. Vorrangdaten entgegengenommen werden, anschließend kann in D1 der (Vorrang-)Datenfluß aktiv gestoppt und in D 4 wieder freigegeben werden. In D3 ist keine Flußregelung möglich.

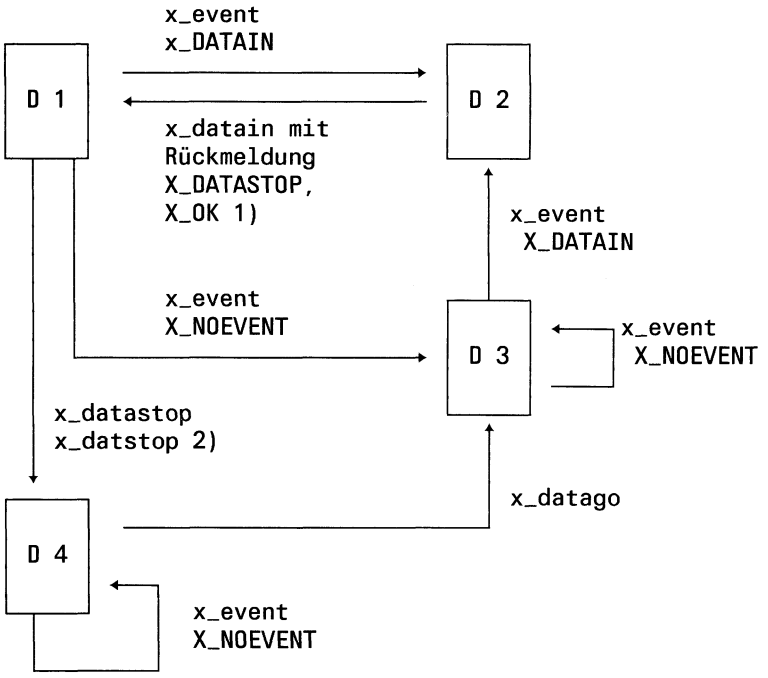


Bild 6-11 Datenempfangsautomat für Normaldaten

1) Bei Rückmeldung X_DATASTOP geht der zugehörige Datensendeautomat in den Zustand D2 über.
2) x_xdatstop ist nur erlaubt, wenn der Datenempfangsautomat für Vorrangdaten in D1 ist.

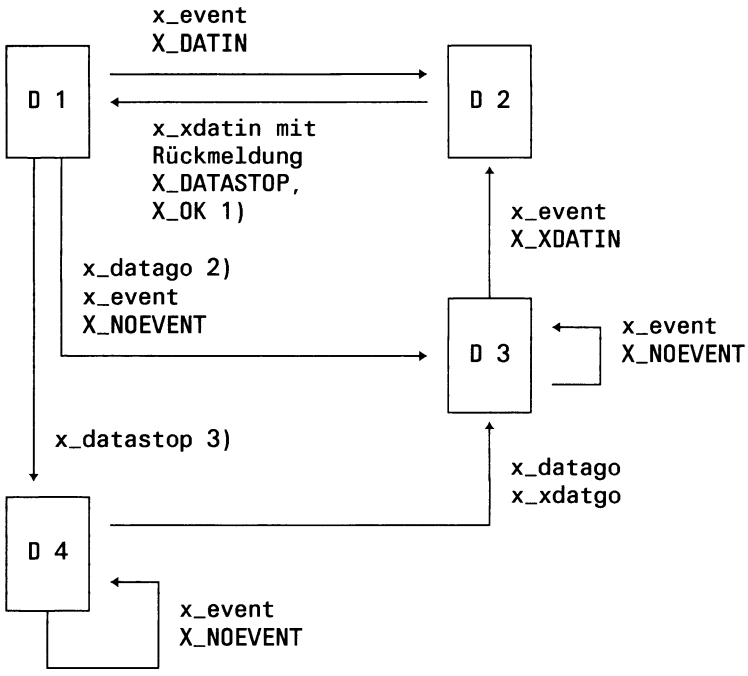


Bild 6-12 Datenempfangsautomat für Vorrangdaten

1) Bei Rückmeldung X_DATASTOP geht der zugehörige Datensendeautomat in den Zustand D2 über.
2) x_datago ist nur erlaubt, wenn der Datenempfangsautomat für Normaldaten in D4 ist.
3) x_xdatstop ist nur erlaubt, wenn der Datenempfangsautomat für Normaldaten in D1 ist.

Programmierhinweise

Das Hauptziel von ICMX(NEA) 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(NEA) unterstützt die Unabhängigkeit für solche TS-Anwendungen, die den folgenden Regeln genügen:

1) Die Anwendung sollte keine expliziten Annahmen über die Länge einer Dateneinheit oder dazu, wie die Dateneinheiten zur Kommunikation verpackt werden, machen.

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

3) Die Adressierung sollte die TS-Anwendung ausschließlich mit Hilfe des TNS behandeln, sie sollte keine physischen Transportadressen in den Programmen aufbauen.

4) NEABX-Funktionen sollten nicht in Signalbehandlungsroutinen aufgerufen werden. Die Signalbehandlung ist nicht dazu geschaffen, asynchrone Verarbeitung 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

```
x_attach();
x_conrq();
for (;;) {
    switch(x_event()) {
    case X_CONCF:
        x_concf();
          :
          :
        x_datarq();
          :
          :
    case X_DATAIN:
        x_datain();
          :
    case X_DISIN:
        x_disin();
        x_datach();
          :
    case X_NOEVENT:
        continue;

    case X_ERROR:
        x_detach();
        exit();
    default:
        :
}
```

gerufene TS-Anwendung

```
x_attach();

for (;;) {
    switch (x_event()) {
    case X_CONIN:
        x_conin();
        x_conrs();
          :
    case X_DATAIN:
        x_datain();
          :
        x_datarq();
          :
          :
    case X_DISIN:
        x_disin();
        x_detach();
          :
    case X_NOEVENT:
        continue;

    case X_ERROR:
        x_detach();
        exit();
    default:
        :
}
```

Konventionen

Bei Verwendung von ICMX(NEA) sind folgende Konventionen einzuhalten:

1) Alle Identifier, die mit '_' beginnen, sind reserviert für die Systemsoftware.

2) Alle Identifier, die mit "t_", "x_", "ts", "Ts", "cmx" oder "neabx" beginnen, sind für NEABX reserviert.

3) Alle Päprozessordefinitionen, die mit "T_", "X_" oder "TS" beginnen, sind für NEABX reserviert.

4) Das Signal SIGTERM wird von den NEABX-Komponenten im Kernel verschickt und in der NEABX-Bibliothek eingefangen (Signalbehandlungsroutine); es sollte nicht für andere Zwecke verwendet werden.

NEABX-Funktionsaufrufe

Die folgenden Seiten beschreiben die NEABX-Aufrufe im Detail. Parameter, in denen NEABX einen vom Aufrufer bereitgestellten Wert erwartet, sind mit "->" gekenzeichnet; Parameter, in denen NEABX nach dem Aufruf einen Wert liefert, sind mit "< " gekennzeichnet; Parameter, in denen der Aufrufer einen Wert bereitstellen muß, der dann von NEABX modifiziert wird, tragen das Kennzeichen "< >".

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

Anmelden bei NEABX - attach process

Mit x_attach() melden Sie den laufenden Prozeß bei NEABX an. Dabei ist ein LOKALER NAME der TS-Anwendung als Parameter anzugeben. Den LOKALEN NAMEN liefert der Transport-Name-Service als eine der Eigenschaften zum GLOBALEN NAMEN einer TS-Anwendung.

Ist dieser Prozeß der erste, der sich mit diesem LOKALEN NAMEN anmeldet, entsteht eine TS-Anwendung. Weitere Prozesse können sich unter demselben LOKALEN NAMEN anmelden und gehören dann zur selben TS-Anwendung.

x_attach() können Sie auch mehrmals im gleichen Prozeß mit verschiedenen LOKALEN NAMEN aufrufen. Dann gehört der Prozeß zu verschiedenen TS-Anwendungen. Wollen Sie den gleichen Prozeß für die Kommunikationsschnittstelle ICMX(L) bei CMX anmelden, so müssen Sie x_attach() und t_attach mit verschiedenen LOKALEN NAMEN aufrufen.

Mit x_attach() legen Sie anwendungsspezifische Parameter für den Prozeß fest:

- den LOKALEN NAMEN,
- welche Arten des Verbindungsaufbaus für den Prozeß in dieser TS-Anwendung möglich sind,
- die Anzahl der Verbindungen, die der Prozeß in dieser TS-Anwendung gleichzeitig haben darf.

x_attach() muß immer der erste NEABX-Aufruf eines Prozesses sein. NEABX kann erst nach dem ersten erfolgreichen x_attach()-Aufruf für eine TS-Anwendung Verbindungswünsche entgegennehmen.

```
int x_attach(name,x_opt)
```

Parameter

-> union x_address *name

Zeiger auf eine Variante, in der der LOKALE NAME der TS-Anwendung anzugeben ist. Der LOKALE NAME ist in der Datei neabx.h als Struktur x_myname definiert. Für die Versorgung dieser Struktur mit den auf das jeweilige CCP abgestimmten Angaben stellt der Transport-Name-Service die Funktion ts13_read_properties bereit. Die LOKALEN NAMEN werden im Directory System mit der Eigenschaft TS_LNAME abgelegt.
Der Aufruf der Funktion erfordert die Bereitstellung einer Struktur Ts_P13, die in tnsx.h definiert und beschrieben wird. Die Komponente ts13prop->ts_name der Struktur Ts_P13 muß gleich TS_LNAME gesetzt werden, um LOKALE NAMEN zu erhalten.

-> union {struct x_opta1 opta1;} *x_opt

Zeiger auf eine Variante, in der eine Struktur x_opta1 oder NULL anzugeben ist. Wenn Sie NULL angeben, setzt CMX die angegebenen Standardwerte. Bei falsch versorgten Parametern ist das Ergebnis X_ERROR (x_error() liefert den Fehlercode X_WPARAMETER). Die Struktur x_opta1 ist in der Datei neabx.h definiert.

```
  struct x_opta1 {
->        int     x_optnr;        /* Options-Nr. */
->        int     x_apmode;       /* Prozeß-Mode */
->        int     x_conlim;       /* Verbindungsanzahl */
  };
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTA1.

x_apmode
legt fest, welche Arten des Verbingungsaufbaus diesem Prozeß in dieser TS-Anwendung möglich sind. Folgende Werte können Sie angeben:

X_ACTIVE	falls der Prozeß aktiv Verbindungen aufbauen soll
X_PASSIVE	falls der Prozeß passiv auf Verbindungswünsche warten soll.

X_REDIRECT falls der Prozeß umgelenkte Verbindungen annehmen soll.

Diese Werte können Sie mit bitweisem 'oder' kombinieren, z.B. X_ACTIVE | X_PASSIVE.

Standardwert: X_ACTIVE | X_PASSIVE | X_REDIRECT.

x_conlim
Maximalzahl der Verbindungen, die dieser Prozeß gleichzeitig haben darf.
Die Maximalzahl ist bei CMX-Aufruf t_attach() definiert.

Ergebnis

X_OK
Der Aufruf war erfolgreich. Der Prozeß hat sich als erster mit diesem LOKALEN NAMEN angemeldet.

X_NOTFIRST
Der Aufruf war erfolgreich. Der Prozeß hat sich als weiterer Prozeß mit diesem LOKALEN NAMEN angemeldet.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Hinweise

1. Beim ersten x_attach() wird im laufenden Prozeß eine Dateikennzahl [CES] belegt und eine Behandlungsfunktion zum Abfangen des Signals SIGTERM [CES] angemeldet. Bei Rechnerkopplung bleibt während der Lebensdauer des Prozesses die Dateikennzahl belegt und die Signalbehandlung angemeldet. Bei Stationskopplung wird beim letzten x_detach() des Prozesses die Dateikennzahl freigegeben und die Behandlung von SIGTERM auf SIG_IGN [CES] gesetzt.

2. Verbindungsanzeigen (ankommende Rufe) stellt NEABX zunächst demjenigen Prozeß einer TS-Anwendung zu, der sich als erster mit X_PASSIVE angemeldet hat. Ist die angegebene Verbindungsgrenze x_conlim erreicht, werden die Verbindungsanzeigen weiteren Prozessen der Anwendung zugestellt. In welcher Reihenfolge die einzelnen Prozesse die Verbindungsanzeigen zugestellt bekommen, ist nicht definiert (weil abhängig von der Implementierung).

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
  .
  .
char    myname[] = "MX2 III-15";                    /* Anwendername */

struct x_opta1 opta1 = { X_OPTA1, X_ACTIVE, 1};  /* Prozeßparameter */

struct x_myname myaddress;                          /* LOKALER NAME */

Ts_head      ts_head = { TS_1VER01 };               /* Versionsnummer */

Ts_leaf_name ts_myname = { 1,                       /* Anzahl der Namensteile */
                           TS_PN,                   /* Typ des Namensteils */
                           sizeof (myname) - 1,     /* Länge des Wertes */
                           myname };                /* Wert des Namensteils */

Ts_property  ts_mprop[] = { { TS_LNAME,             /* Name der Eigenschaft */
                              TS_ITEM },            /* Typ der Eigenschaft */
                            { TS_EMPTYPROP } };     /* Listenende-Kennzeichen */

Ts_P13 ts13m = { &ts_head,                  /* Standardkopf */
                 TS_CMX_DIR_ID,             /* Directory-Identifier */
                 &ts_myname,                /* Objektname */
                 NULL,                      /* nicht unterstützt */
                 (char *) &myaddress,       /* Werteliste für Eigenschaften */
                 sizeof(myaddress),         /* Länge der Werteliste */
                 ts_mprop,                  /* Liste verlangter Eigenschaften */
                 NULL, NULL, NULL };        /* nicht unterstützt */
   .
   .
if (ts13_read_properties(&ts13m) == TS_ERROR) {
    fprintf (stderr, "ts13_read_properties: Fehler %d (Typ: %d Kl.: %d)\n",
        ts_head.ts_errvalue, ts_head.ts_retcode, ts_head.ts_errclass);
    exit ();
}
else if (ts_mprop[0].ts_length == 0) {
    fprintf (stderr, "ts13_read_properties: nichts gefunden\n");
    exit ();
}

if (x_attach(&myaddress, &opta1) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_attach\n", x_error());
```

Der Aufruf meldet den Prozeß unter dem lokal eindeutigen Namen "MX2 III-15" bei NEABX an. Der Prozeß soll nur aktiv Verbindungen aufbauen können, es soll nur eine Verbindung bestehen können.

Verbindung herstellen - connection confirmation

Mit x_concf() nehmen Sie das Ereignis X_CONCF entgegen. X_CONCF erhalten Sie angezeigt, wenn die gerufene TS-Anwendung Ihre Verbindungsanforderung bestätigt hat. Mit x_concf() übergibt NEABX die Informationen, die die gerufene TS-Anwendung bei der Antwort mitgegeben hat.
Wenn der Aufruf x_concf() erfolgreich war, ist die Verbindung fertig aufgebaut. Meldet NEABX das Ergebnis X_REPEAT, so müssen Sie x_concf mit gleichen Parametern noch einmal aufrufen, sobald x_event() das Ereignis X_REPCCF anzeigt.

```
int x_concf(tref,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz ein, die Sie erhalten, wenn der Aufruf von x_event() das Ereignis X_CONCF meldet.

<- union {struct x_optc1 optc1;} *x_opt
Zeiger auf eine Variante, in die NEABX eine Struktur x_optc1 einträgt. Mit dieser Struktur können Sie die Information abfragen, die die gerufene TS-Anwendung bei ihrer Antwort mitgeliefert hat.
Die Struktur x_optc1 ist in der Datei neabx.h definiert.

```
  struct x_optc1 {
->        int     x_optnr;        /* Options-Nr. */
<-        char    *x_udatap;      /* Datenpuffer */
<>        int     x_udatal;       /* Länge des Datenpuffers */
<-        int     x_xdata;        /* Vorrangdaten-Auswahl */
<-        int     x_timeout;      /* Inaktiv-Zeit */
<-        char    x_passwd[4];    /* Verbindungspasswort */
<-        int     x_prot;         /* Protokoll Datenphase */
  } ;
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTC1.

x_udatap
Zeiger auf einen Datenbereich. In diesen Bereich trägt NEABX die Benutzerverbindungsnachricht der gerufenen TS-Anwendung ein. Den Bereich müssen Sie so groß wählen, daß die Verbindungsnachricht hineinpaßt. Die Verbindungsnachricht ist maximal X_MSG_SIZE Byte lang. Ist der Datenbereich kleiner als die Länge der empfangenen Verbindungsnachricht, ist das Ergebnis X_ERROR.
Die Benutzerverbindungsnachricht besteht aus dem NEABV-Protokoll (siehe Abschnitt 7).
Das NEABV-Protokoll kann mit dem NEABV-Aufruf neavi() analysiert werden.
Die NEABV-Benutzernachricht wird im Code des Partners geliefert.
Das heutige Transportsystem auf BS2000-Seite liefert keine Verbindungsnachrichten, die mit x_concf() zugestellt werden.

x_udatal
Vor dem Aufruf geben Sie als Länge des bereitgestellten Datenbereichs X_MSG_SIZE Byte an. Beim Aufruf trägt NEABX die Länge der empfangenen Verbindungsnachricht ein.

x_xdata
liefert die Antwort der gerufenen TS-Anwendung, ob Vorrangdaten benutzt werden können. Die Antwort ist verbindlich.

X_YES Vorschlag für Vorrangdaten senden und empfangen wird akzeptiert.

X_NO Benutzung von Vorrangdaten wird vom Partner abgelehnt.

x_timeout
Der Inhalt dieses Feldes ist mit NIL belegt.

x_passwd
Verbindungspaßwort. Ankommend wird kein Verbindungspaßwort zugestellt. x_passwd ist mit NIL belegt. Dies geschieht in Anlehnung an die ISO-Norm.

x_prot
bestimmt, ob in der Datenphase mit NEABX-Protokoll gearbeitet werden soll:

X_NEABX	In der Datenphase wird immer ein NEABX-Protokoll gesendet und erwartet.
X_NOBX	In der Datenphase wird ohne NEABX-Protokoll gearbeitet.

Von heutigen BS2000-Anwendungen (DCAM, TIAM, UTM) ist X_NEABX zu erwarten.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_REPEAT
Der Aufruf war erfolgreich. Wenn x_event() das Ereignis X_REPCCF anzeigt, muß x_concf() nochmals aufgerufen werden.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>

#define         TRUE    1
#define         FALSE   0
  .
  .
int     debug = 1;                              /* mehr Information */
int     rc;                                     /* Returncode */
int     tref;                                   /* Transportreferenz */
int     repccf = FALSE;                         /* Wiederholanzeige */

struct x_optc1 optc1;                           /* Option Verbindungsphase */
  .
  .
switch (x_event(&tref, X_WAIT, NULL)) {
case X_CONCF:
    if ((rc = x_concf(&tref, &optc1)) == X_ERROR) {
        fprintf(stderr, "Fehler 0x%x bei x_concf für tref %x\n",
            x_error(), tref);
        exit ();
    }
    else
        repccf = (rc == X_REPEAT) ? TRUE : FALSE;
    if (debug)
        fprintf(stderr, "Verbindung aufgebaut, tref %x\n", tref);
    break;
   .
   .
}
```

Der Prozeß wartet auf das nächste Ereignis. Wenn eine gerufene TS-Anwendung eine Verbindungsanforderung dieses Prozesses bestätigt (Ereignis X_CONCF), so liefert NEABX die Transportreferenz der angeforderten Verbindung. Liefert x_concf() das Ergebnis X_REPEAT, wird der Aufruf wiederholt, wenn x_event() das Ereignis X_REPEAT anzeigt. Mit dem erfolgreichen Aufruf von x_concf() steht die Verbindung.

Verbindungsanforderung entgegennehmen - connection indication

Mit x_conin() nehmen Sie die Verbindungsanforderung einer anderen TS-Anwendung entgegen. x_conin() rufen Sie auf, wenn x_event() das Ereignis X_CONIN meldet. Der Aufruf liefert

- die TRANSPORTADRESSE der rufenden und den LOKALEN NAMEN der lokalen TS-Anwendung,
- die Benutzerdaten, die die rufende TS-Anwendung beim Aufruf x_conrq() mitgegeben hat.

Anschließend können Sie die Verbindungsanforderung mit x_conrs() bestätigen oder mit x_disrq() ablehnen.

Meldet NEABX das Ergebnis X_REPEAT, so müssen Sie x_conin() mit gleichen Parametern noch einmal aufrufen, sobald x_event() das Ereignis X_REPCIN anzeigt.

```
int x_conin(tref,toaddr,fromaddr,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz ein, die Sie erhalten, wenn der Aufruf von x_event() das Ereignis X_CONIN meldet.

<- union x_address *toaddr
Zeiger auf eine Variante, in die NEABX den LOKALEN NAMEN der gerufenen lokalen TS-Anwendung einträgt. Diese Information ist dann wichtig, wenn ein Prozeß mehrere TS-Anwendungen steuert. Sie erfahren dadurch, welche TS-Anwendung die Verbindung bekommen soll. Der LOKALE NAME ist in der Datei neabx.h als Struktur x_myname definiert. Mit der Funktion ts11_read_leafs kann der Transport-Name-Service zum LOKALEN NAMEN den GLOBALEN NAMEN der gerufenen lokalen TS-Anwendung ermitteln. Die LOKALEN NAMEN werden im Directory-System mit der Eigenschaft TS_LNAME abgelegt.
Der Aufruf der Funktion erfordert die Bereitstellung einer Struktur Ts_P11, die in tnsx.h definiert und beschrieben wird. Die Komponente ts11filter->ts_pr_name der Struktur Ts_P11 muß gleich TS_LNAME gesetzt werden, um zu LOKALEN NAMEN den GLOBALEN NAMEN zu erhalten.

<- union x_address *fromaddr
Zeiger auf eine Variante, in die NEABX die TRANSPORTADRESSE der rufenden TS-Anwendung einträgt.
Für die Adresse ist in der Datei neabx.h die Struktur x_partaddr definiert. Analog zur Struktur x_myname kann mit Hilfe der Funktion ts11_read_leafs zur TRANSPORTADRESSE der GLOBALE NAME der rufenden TS-Anwendung ermittelt werden. Die TRANSPORT-ADRESSEN werden im Directory-System mit der Eigenschaft TS_TRANS abgelegt.
Der Aufruf der Funktion erfordert die Bereitstellung einer Struktur Ts_P11, die in tnsx.h definiert und beschrieben wird. Die Komponente ts11filter->ts_pr_name der Struktur Ts_P11 muß gleich TS_TRANS gesetzt werden, um zu TRANSPORTADRESSEN den GLOBALEN NAMEN zu erhalten.

<- union {struct x_optc1 optc1;} *x_opt

Zeiger auf eine Variante, in die NEABX eine Struktur x_optc1 einträgt.
Mit dieser Struktur können Sie die Information abfragen, die die rufende TS-Anwendung bei der Verbindungsanforderung mitgeliefert hat.
Die Struktur x_optc1 ist in der Datei neabx.h definiert.

```
  struct x_optc1 {
->        int     x_optnr;        /* Options-Nr. */
<-        char    *x_udatap;      /* Datenpuffer */
<>        int     x_udatal;       /* Länge des Datenp. */
<-        int     x_xdata;        /* Vorrangdaten-Auswahl */
->        int     x_timeout;      /* Inaktiv-Zeit */
<-        char    x_passwd[4];    /* Verbindungspasswort */
<-        int     x_prot;         /* Protokoll Datenphase */
  } ;
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTC1.

x_udatap
Zeiger auf einen Datenbereich. In diesen Bereich trägt NEABX die Benutzerverbindungsnachricht der rufenden TS-Anwendung ein. Den Bereich müssen Sie so groß wählen, daß die Verbindungsnachricht hineinpaßt. Die Verbindungsnachricht ist maximal X_MSG_SIZE Byte lang.
Ist der Datenbereich kleiner als die Länge der empfangenen Verbindungsnachricht, ist das Ergebnis X_ERROR.

Die Benutzerverbindungsnachricht besteht aus dem NEABV-Protokoll (siehe Abschnitt 7).
Das NEABV-Protokoll kann mit dem NEABV-Aufruf neavi() analysiert werden.

x_udatal
Vor dem Aufruf geben Sie als Länge des bereitgestellten Datenbereichs X_MSG_SIZE Byte an. Beim Aufruf trägt NEABX die Länge der empfangenen Benutzerverbindungsnachricht ein.

x_xdata
teilt mit, ob für die Verbindung die Benutzung von Vorrangdaten vorgeschlagen wird:

X_YES Vorrangdaten senden und empfangen wird vorgeschlagen.

X_NO Keine Vorrangdaten zu senden wird vorgeschlagen.

x_timeout
Zeitüberwachung für die Verbindung. Sie können folgende Werte angeben:

X_NO keine Zeitüberwachung.

n n Sekunden darf die Verbindung inaktiv sein. Danach baut NEABX die Verbindung ab.

x_timeout gilt nur für die lokale Anwendung. NEABX teilt den Wert der rufenden TS-Anwendung nicht mit.

x_passwd
Verbindungspaßwort. Ankommend wird kein Verbindungspaßwort zugestellt. x_passwd ist mit NIL belegt. Dies geschieht in Anlehnung an die ISO-Norm.

x_prot
enthält den Vorschlag des Partners, ob in der Datenphase mit NEABX-Protokoll gearbeitet werden soll:

X_NEABX In der Datenphase wird immer ein NEABX-Protokoll gesendet und erwartet.

X_NOBX In der Datenphase wird ohne NEABX-Protokoll gearbeitet.

Von heutigen BS2000-Anwendungen (DCAM, TIAM, UTM) ist X_NEABX zu erwarten.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_REPEAT
Der Aufruf war erfolgreich. Wenn x_event() das Ereignis X_REPCIN anzeigt, muß x_conin() wiederholt werden.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tnsx.h>
#include        <stdio.h>

#define         TRUE    1
#define         FALSE   0
 .
 .
int     debug = 1;                              /* mehr Information */
int     rc;                                     /* Returncode */
int     tref;                                   /* Transportreferenz */
int     repcin = FALSE;                         /* Wiederholanzeige */

struct x_optc1 optc1;                           /* Option Verbindungsphase */

struct x_myname    myaddress;                   /* LOKALER NAME */
union  x_partaddr  partaddr;                    /* TRANSPORTADRESSE */

Ts_head      ts_head = { TS_1VER01 };           /* Versionsnummer */

Ts_leaf_name ts_sname = { 0 };                  /* Anzahl der Namensteile */

Ts_leaf_name ts_pname;                          /* Anwendername */

Ts_filter    ts_filter = { TS_TRANS,            /* Name der Eigenschaft */
                           NULL,                /* vorläufiger Wert */
                           (char *)&partaddr }; /* Wert der Eigenschaft */

char  ts_nlist[TS_LPN];

Ts_P11 ts11p = { &ts_head,              /* Standardkopf */
                 TS_CMX_DIR_ID,         /* Directory-Identifier */
                 &ts_sname,             /* Suchkriterium (Name) */
                 TS_GENERAL,            /* Suchmodus */
                 &ts_filter,            /* Suchkriterium (Eigenschaft) */
                 TS_LOCAL,              /* Suchmodus */
                 NULL, NULL,            /* nicht unterstützt */
                 ts_nlist,              /* Werteliste für Namensteile */
```

```
                sizeof (ts_nlist),      /* Länge der Werteliste */
                1,                      /* Anzahl der erwarteten Namen */
                ts_pname,               /* Liste für gefundene Objekte */
                NULL };                 /* Anzahl gefundener Objekte */
  .
  .
switch (x_event(&tref, X_WAIT, NULL)) {
case X_CONIN:
if((rc = x_conin(&tref, &myaddress, &partaddr, &optc1)) == X_ERROR) {
        fprintf(stderr, "Fehler 0x%x bei x_conin für tref %x\n",
            x_error(), tref);
        exit ();
    }
    else
        repcin = (rc == X_REPEAT) ? TRUE : FALSE;

    ts_filter.ts_pr_length = partaddr.x_partaddr.x_palng;
    if (ts11_read_leafs(&ts11p) == TS_ERROR) {
        fprintf (stderr, "ts11_read_leafs: Fehler %d (Typ: %d Kl.: %d)\n",
            ts_head.ts_errvalue, ts_head.ts_retcode, ts_head.ts_errclass);
        exit ();
    }
    else if (ts11p.ts11real_no == 0) {
        fprintf (stderr, "ts11_read_leafs: nichts gefunden\n");
        exit ();
    }
    if (debug)
        fprintf(stderr, "Verbindungwunsch von %s erhalten, tref %x\n",
            ts_pname.ts_value, tref);
    break;
   .
   .
}
```

Der Prozeß wartet auf das nächste Ereignis. Wenn eine rufende TS-Anwendung einen Verbindungsaufbau zu diesem Prozeß anfordert (Ereignis X_CONIN), enthält die Struktur myaddress den LOKALEN NAMEN der gerufenen lokalen TS-Anwendung und die Variante partaddress eine Struktur mit der TRANSPORTADRESSE der rufenden TS-Anwendung. Im Beispiel wird nur der Anwendername der rufenden TS-Anwendung ermittelt.

Verbindung anfordern - connection request

x_conrq() fordert den Aufbau einer Verbindung von der lokalen TS-Anwendung zu einer gerufenen TS-Anwendung an (aktiver Verbindungsaufbau, abgehender Ruf). Der gerufenen TS-Anwendung stellt NEABX das Ereignis X_CONIN zu. Sie können der gerufenen TS-Anwendung mit der Verbindungsanforderung Informationen mitschicken.

Meldet NEABX das Ergebnis X_REPEAT, so müssen Sie x_conrq() mit gleichen Parametern noch einmal aufrufen, sobald x_event() das Ereignis X_REPCRQ anzeigt.

```
int x_conrq(tref,toaddr,fromaddr,x_opt)
```

Parameter

<- int *tref
Zeiger auf die Transportreferenz. Die Transportreferenz wird von NEABX beim ersten Aufruf eingetragen und kennzeichnet diese Verbindung für NEABX eindeutig. Sie ist bei allen Aufrufen anzugeben, die sich auf diese Verbindung beziehen.

-> union x_address *toaddr
Zeiger auf eine Variante, in der die TRANSPORTADRESSE der gerufenen TS-Anwendung anzugeben ist.
Die TRANSPORTADRESSE ist in der Datei neabx.h als Struktur x_partaddr definiert. Für die Versorgung dieser Struktur mit den auf das jeweilige CCP abgestimmten Angaben stellt der Transport-Name-Service die Funktion ts13_read_properties bereit. Die TRANSPORTADRESSEN werden im Directory System mit der Eigenschaft TS_TRANS abgelegt.
Der Aufruf der Funktion erfordert die Bereitstellung einer Struktur Ts_P13, die in tnsx.h definiert und beschrieben wird. Die Komponente ts13prop->ts_name der Struktur Ts_P13 muß gleich TS_TRANS gesetzt werden, um TRANSPORTADRESSEN zu erhalten.

-> union x_address *fromaddr
Zeiger auf eine Variante, in der der LOKALE NAME der rufenden TS-Anwendung anzugeben ist.
Der LOKALE NAME ist in der Datei neabx.h als Struktur x_myname definiert. Die Versorgung der Struktur x_myname erfolgt analog zur Struktur x_partaddr mit der Funktion ts13_read_properties.

Die LOKALEN NAMEN werden im Directory System mit der Eigenschaft TS_LNAME abgelegt. Die Komponente ts13prop->ts_name der Struktur Ts_P13 muß gleich TS_LNAME gesetzt werden, um LOKALE NAMEN zu erhalten.
Abgesehen vom Wiederholungsfall ist derselbe LOKALE NAME anzugeben wie beim Aufruf x_attach() für diese TS-Anwendung.

-> union {struct x_optc1 optc1;} *x_opt
Zeiger auf eine Variante, in der eine Struktur x_optc1 anzugeben ist. Bei falsch versorgten Parametern ist das Ergebnis X_ERROR. x_error() liefert den Fehlercode X_WPARAMETER. Mit dieser Struktur können Sie der gerufenen TS-Anwendung Informationen mitschicken. Diese erhält die Daten mit der Entgegennahme der Verbindungsanforderung.
Die Struktur x_optc1 ist in der Datei neabx.h definiert.

```
  struct x_optc1 {
->          int     x_optnr;        /* Options-Nr. */
->          char    *x_udatap;      /* Datenpuffer */
->          int     x_udatal;       /* Länge des Datenp. */
->          int     x_xdata;        /* Vorrangdatenauswahl */
->          int     x_timeout;      /* Inaktiv-Zeit */
->          char    x_passwd[4];    /* Verbindungspasswort */
->          int     x_prot;         /* Protokoll Datenphase */
  } ;
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTC1.

x_udatap

Zeiger auf einen Bereich mit Daten, die NEABX an die gerufene TS-Anwendung übergibt.
Er enthält die Benutzerverbindungsnachricht, die in diesem Bereich linksbündig stehen muß. Anschließend enthält er 8 Byte Reserve als Platzreservierung für das NEABX-Protokoll, das CMX(NEA) hinzufügt.
Die Benutzerverbindungsnachricht besteht aus dem NEABV-Protokoll (siehe Abschnitt 7).
Das NEABV-Protokoll kann auch mit dem NEABV-Aufruf neavo() für Stations- und Rechnerkopplung erzeugt werden. Dieser liefert die richtige Länge für x_udatal zurück, die hier direkt eingesetzt werden kann.
Die NEABV-Benutzernachricht erwartet NEABX im Fall der Stationskopplung in ASCII.

x_udatal

Länge des Bereichs x_udatap, der von NEABX übergeben werden soll. Die Länge ist die Summe der Benutzerverbindungsnachricht und der Reserve für das NEABX-Protokoll Die Angabe muß mit der tatsächlichen Belegung von x_udatap (einschließlich Platzreservierung) übereinstimmen, sonst ist das Ergebnis X_ERROR.
Maximale Länge: X_MSG_SIZE.
Minimale Länge: Bei Rechnerkopplung 20 Byte, bei Stationskopplung 13 Byte.

x_xdata

schlägt der gerufenen TS-Anwendung vor, die Benutzung von Vorrangdaten zu erlauben, oder schließt sie aus:

X_YES Vorrangdaten senden und empfangen wird vorgeschlagen.
Bei Rechnerkopplung ist für die Kommunikation mit heutigen BS2000-Anwendungen über TIAM, DCAM, UTM immer X_YES anzugeben.

X_NO Benutzung von Vorrangdaten wird nicht eingeräumt.
Bei Stationskopplung ist immer X_NO anzugeben.

x_timeout

Zeitüberwachung für die Verbindung. Sie können folgende Werte angeben:

X_NO keine Zeitüberwachung.

n n Sekunden darf die Verbindung inaktiv sein. Danach baut NEABX die Verbindung ab.

x_timeout gilt nur für die lokale Anwendung. NEABX teilt der rufenden TS-Anwendung den Wert nicht mit. Bei Stationskopplung wird der Parameter x_timeout ignoriert, da keine Zeitüberwachung möglich ist.

x_passwd
Verbindungspaßwort. Für Partneranwendungen, die dies benötigen, geben Sie eine vier Byte lange Binärinformation an. Wollen Sie kein Verbindungspaßwort mitgeben, belegen Sie x_passwd mit NIL. Dies sollte angelehnt an die ISO-Norm bei neuen Anwendungen geschehen.

x_prot
bestimmt, ob in der Datenphase mit NEABX-Protokoll gearbeitet werden soll.
Bei Verbindungen zu heutigen BS2000-Anwendungen (DCAM, TIAM, UTM) ist immer X_NEABX anzugeben.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_REPEAT
Der Aufruf war erfolgreich. Wenn x_event() das Ereignis X_REPCRQ anzeigt, muß x_conrq() nochmals aufgerufen werden.

X_DATASTOP
Der Aufruf war erfolgreich. In einer folgenden Datenphase muß das Ereignis X_DATAGO abgewartet werden.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Hinweis

Beim Aufruf von x_conrq() wird überprüft, ob die rufende TS-Anwendung und die gerufene TS-Anwendung mit NEABX-Service arbeiten. Andernfalls ist das Ergebnis X_ERROR (x_error() liefert den Fehlercode X_NOTNEABX).

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tnsx.h>
#include        <stdio.h>

#define         TRUE    1
#define         FALSE   0
 .
 .
int     rc;                                     /* Returncode */
int     datastop = FALSE;                       /* Indikator Datenfluß */
char    partname[] = "MX2 IV-1";                /* Anwendername Partner */
int     repcrq = FALSE;                         /* Wiederholanzeige */
char    udatap[]="<Benutzerdaten>";             /* Datenpuffer */

struct x_optc1 optc1 = { X_OPTC1,               /* Options-Nr. */
                         &udatap[0],            /* Datenpuffer */
                         sizeof(udatap),        /* Länge des Datenpuffers */
                         X_NO,                  /* Vorrangdatenauswahl */
                         60,                    /* Inaktiv-Zeit */
                         0X2C4F502C,            /* Verbindungspassword */
                         X_NOBX };              /* Protokoll Datenphase */

struct x_myname    myaddress;                   /* LOKALER NAME */
union  x_partaddr  partaddr;                    /* TRANSPORTADRESSE */

Ts_head      ts_head = { TS_1VER01 };           /* Versionsnummer */

Ts_leaf_name ts_pname = { 1,                    /* Anzahl der Namensteile */
                          TS_PN,                /* Typ des Namensteils */
                          sizeof(partname) - 1, /* Länge des Wertes */
                          partname };           /* Wert des Namensteils */

Ts_property  ts_pprop[] = { { TS_TRANS,         /* Name der Eigenschaft */
                              TS_ITEM },        /* Typ der Eigenschaft */
                            { TS_EMPTYPROP } }; /* Listenende-Kennzeichen */

Ts_P13 ts13p = { &ts_head,              /* Standardkopf */
                 TS_CMX_DIR_ID,         /* Directory-Identifier */
                 &ts_pname,             /* Objektname */
                 NULL,                  /* nicht unterstützt */
                 (char *)&partaddr,     /* Werteliste für Eigenschaften */
                 sizeof (partaddr),     /* Länge der Werteliste */
                 ts_pprop,              /* Liste verlangter Eigenschaften */
                 NULL, NULL, NULL };    /* nicht unterstützt */
   .
   .
if (ts13_read_properties(&ts13p) == TS_ERROR) {
    fprintf (stderr, "ts13_read_properties: Fehler %d (Typ: %d Kl.: %d)\n",
        ts_head.ts_errvalue, ts_head.ts_retcode, ts_head.ts_errclass);
    exit ();
}
else if (ts_pprop[0].ts_length == 0) {
    fprintf (stderr, "ts13_read_properties: nichts gefunden\n");
    exit ();
```

```
}

if ((rc = x_conrq(&tref, &partaddr, &myaddress, &optc1)) == X_ERROR) {
    fprintf(stderr, "Fehler 0x%x bei x_conrq für tref %x\n",
        x_error(), tref);
    exit ();
}
else
    repcrq = (rc == X_REPEAT) ? TRUE : FALSE;
if (rc == X_DATASTOP)
    datastop = TRUE;
    .
    .
```

Die erforderlichen Vereinbarungen und Aufrufe für myaddress erfolgen wie im Beispiel zu x_attach().
Der Aufruf fordert eine Verbindung zur Anwendung "MX2 IV-1" an. Es wird eine benutzerspezifische Information beim Verbindungsaufbau mitgeschickt. In tref liefert NEABX die Transportreferenz der angeforderten Verbindung.

Verbindungsanforderung bestätigen - connection response

Mit x_conrs() bestätigen Sie die Verbindungsanforderung einer rufenden TS-Anwendung. NEABX stellt der rufenden TS-Anwendung diese Antwort mit dem Ereignis X_CONCF zu. Mit der Bestätigung können Sie der rufenden TS-Anwendung Informationen mitschicken.

Bevor Sie x_conrs() aufrufen, muß NEABX das Ereignis X_CONIN gemeldet haben. Sie müssen dieses Ereignis mit x_conin() entgegennehmen (passiver Verbindungsaufbau). Wenn der Aufruf erfolgreich ist, ist die Verbindung für den laufenden Prozeß fertig aufgebaut.

```
int x_conrs(tref,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz ein, die Sie erhalten, wenn der Aufruf von x_event() das Ereignis X_CONIN meldet.

-> union {struct x_optc1 optc1;} *x_opt
Zeiger auf eine Variante, in der eine Struktur x_optc1 anzugeben ist. Bei falsch versorgten Parametern ist das Ergebnis X_ERROR (x_error() liefert den Fehlercode X_WPARAMETER).
Mit dieser Struktur können Sie der rufenden TS-Anwendung bei der Antwort Informationen mitschicken.
Die Struktur x_optc1 ist in der Datei neabx.h definiert.

```
  struct x_optc1 {
->        int     x_optnr;        /* Options-Nr. */
->        char    *x_udatap;      /* Datenpuffer */
->        int     x_udatal;       /* Länge des Datenp. */
->        int     x_xdata;        /* Vorrangdatenauswahl */
          int     x_timeout;      /* Inaktiv-Zeit */
->        char    x_passwd[4];    /* Verbindungspasswort */
->        int     x_prot;         /* Protokoll Datenphase */
  } ;
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTC1.

x_udatap
Zeiger auf einen Bereich mit Daten, die NEABX an die gerufene TS-Anwendung übergibt.
Er enthält die Benutzerverbindungsnachricht, die in diesem Bereich linksbündig stehen muß. Anschließend enthält er 8 Byte Reserve als Platzreservierung für das NEABX-Protokoll, das CMX(NEA) hinzufügt.
Die Benutzerverbindungsnachricht besteht aus dem NEABV-Protokoll (siehe Abschnitt 7).
Das NEABV-Protokoll kann auch mit dem NEABV-Aufruf neavo() für Stations- und Rechnerkopplung erzeugt werden. Dieser liefert die richtige Länge für x_udatal zurück, die hier direkt eingesetzt werden kann.

x_udatal
Länge des Bereichs x_udatap, der von NEABX übergeben werden soll. Die Länge ist die Summe der Benutzerverbindungsnachricht und der Reserve für das NEABX-Protokoll. Die Angabe muß mit der tatsächlichen Belegung von x_udatap (einschließlich Platzreservierung) übereinstimmen, sonst ist das Ergebnis X_ERROR.
Maximale Länge: X_MSG_SIZE Byte.
Minimale Länge: Bei Rechnerkopplung 20 Byte, bei Stationskopplung 13 Byte.

x_xdata
antwortet auf den Vorschlag der rufenden TS-Anwendung, ob Vorrangdaten benutzt werden können. Die Antwort ist verbindlich. Auf den Vorschlag X_NO der rufenden TS-Anwendung darf nur mit X_NO geantwortet werden.

X_YES Vorschlag für Vorrangdaten senden und empfangen wird akzeptiert.
Bei Rechnerkopplung ist für die Kommunikation mit heutigen BS2000-Anwendungen über TIAM, DCAM, UTM i.a. X_YES anzugeben.

X_NO Benutzung von Vorrangdaten wird abgelehnt.
Bei Stationskopplung ist immer X_NO anzugeben.

x_timeout
Der Inhalt dieses Feldes ist nicht relevant.

x_passwd
Verbindungspaßwort. Sie können eine vier Byte lange Binärinformation angeben. Wollen Sie kein Verbindungspaßwort mitgeben, belegen Sie x_passwd mit NIL.

x_prot
Bei Verbindungen zu heutigen BS2000-Anwendungen (DCAM, TIAM, UTM) ist immer X_NEABX anzugeben.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_DATASTOP
Der Aufruf war erfolgreich. In einer folgenden Datenphase muß das Ereignis X_DATAGO abgewartet werden.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>

#define         TRUE    1
#define         FALSE   0
  .
  .
int     rc;                                     /* Returncode */
int     datastop = FALSE;                       /* Indikator Datenfluss */
int     debug = 1;                              /* mehr Information */
int     tref;                                   /* Transportreferenz */
int     repcin = FALSE;                         /* Wiederholanzeige */
char    updatap [] = "<Benutzerdaten>";         /* Datenpuffer */

struct x_optc1 optc;                            /* Option Verbindungsphase */

struct x_optc1 optc1 = { X_OPTC1,               /* Options-Nr. */
                         &udatap[0],            /* Datenpuffer */
                         sizeof(udatap),        /* Länge des Datenpuffers */
                         X_NO,                  /* Vorrangdatenauswahl */
                         60,                    /* Inaktiv-Zeit */
                         0X2C4F502C,            /* Verbindungspaßwort */
                         X_NOBX };              /* Protokoll Datenphase */

struct x_myname     myaddress;                  /* LOKALER NAME */
```

```
union  x_partaddr  partaddr;                    /* TRANSPORTADRESSE */
  .
  .
switch (x_event(&tref, X_WAIT, NULL)) {
case X_CONIN:
if ((rc = x_conin(&tref, &myaddress, &partaddr, &optc)) == X_ERROR) {
        fprintf(stderr, "Fehler 0x%x bei x_conin tref %x\n",
            x_error(), tref);
        exit ();
    }
    else
        repcin = (rc == X_REPEAT) ? TRUE : FALSE;
      .
      .
if ((rc = x_conrs(&tref, &optc1)) == X_ERROR) {
        fprintf(stderr, "Fehler 0x%x bei x_conrs tref %x\n", x_error(), tref);
        exit ();
    }
    if (debug)
        fprintf(stderr, "Verbindungwunsch bestätigt, tref %x\n", tref);
    if (rc == X_DATASTOP)
        datastop = TRUE;
    break;
  .
  .
}
```

Der Prozeß wartet auf das nächste Ereignis. Wenn eine rufende TS-Anwendung einen Verbindungsaufbau zu diesem Prozeß wünscht (Ereignis X_CONIN), enthält tref die Transportreferenz der angeforderten Verbindung.
Durch den Aufruf von x_conrs() mit dieser Transportreferenz wird der Verbindungswunsch bestätigt. Es wird keine Verbindungsnachricht mitgeschickt, doch wird die Benutzung von Vorrangdaten ausgeschlossen und ein Verbindungspaßwort vereinbart. Die Datenphase wird ohne NEABX-Protokoll abgewickelt. NEABX baut die Verbindung ab, wenn sie 60 Sekunden lang inaktiv ist.

Datenfluß freigeben - datago

x_datago() gibt den gesperrten Datenfluß auf der angegebenen Verbindung frei. Der laufende Prozeß teilt NEABX mit, daß er wieder bereit ist, Daten zu empfangen. Wenn die Benutzung von Vorrangdaten vereinbart ist, wird auch der Vorrangdatenfluß freigegeben, falls er gesperrt war. Der Aufruf bewirkt im einzelnen:

- Die lokale TS-Anwendung erhält die Ereignisse X_DATAIN und X_XDATIN zugestellt, falls sie anstehen.
- Die ferne TS-Anwendung erhält das Ereignis X_DATAGO zugestellt. Sie darf wieder Daten senden.

```
int x_datago(tref)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, für die Sie den Datenfluß freigeben wollen.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beipiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
    .
int   tref;       /* Transportreferenz */
    .
if (x_datago(&tref) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_datago für tref %x\n",
        x_error(), tref);
```

Der Aufruf gibt den Datenfluß für die mit tref angegebene Verbindung frei.

Daten empfangen - data indication

Mit x_datain() nimmt der laufende Prozeß eine von der fernen TS-Anwendung gesendete Dateneinheit entgegen. Zuvor muß NEABX bei x_event() das Ereignis X_DATAIN gemeldet haben. Dabei übergibt NEABX in tref die Transportreferenz der Verbindung, auf der die Nachricht eingegangen ist.

Der Indikator x_chain zeigt an, ob noch weitere Dateneinheiten zur Nachricht gehören. Jede weitere Dateneinheit zeigt NEABX erneut mit dem Ereignis X_DATAIN an.

```
int x_datain(tref,x_datap,x_datal,x_chain,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz ein, die Sie erhalten, wenn der Aufruf von x_event() das Ereignis X_DATAIN meldet.

<- char *x_datap
Zeiger auf einen Bereich, in den NEABX die empfangenen Daten einträgt.
Ist x_opt gleich NULL, so übergibt NEABX die empfangenen Daten der lokalen TS-Anwendung.
Ist x_opt ungleich NULL, so wird das NEABX-Protokoll von NEABX vor der Übergabe an die lokale TS-Anwendung entfernt. Ist kein NEABX-Protokoll vorhanden, ist das Ergebnis X_ERROR.

<- int *x_datal
Vor dem Aufruf geben Sie die Länge des Datenbereichs x_datap an, mindestens aber die Länge einer Dateneinheit, die Sie für jede Transportverbindung mittels x_info() erfragen müssen. Beim Aufruf trägt NEABX die Anzahl der eingetragenen Byte ein, die an die lokale TS-Anwendung übergeben werden.

<- int *x_chain
Zeiger auf einen Indikator, mit dem NEABX anzeigt, ob noch weitere Dateneinheiten zur Nachricht gehören.

Folgende Werte sind möglich:

X_MORE Es sind noch Dateneinheiten der Nachricht vorhanden. Für jede weitere Dateneinheit meldet NEABX ein eigenes Ereignis X_DATAIN.

X_END Es sind keine weiteren Dateneinheiten vorhanden. Die Nachricht ist fertig übertragen.

<- union {struct x_optd1 optd1;} *x_opt
Zeiger auf eine Variante, die die Struktur x_optd1 oder die Angabe NULL enthält. Die Angabe NULL ist obligatorisch, wenn

a) eine weitere Dateneinheit einer Nachricht entgegengenommen werden soll, also die vorhergehende Dateneinheit auf dieser Verbindung mit *x_chain = X_MORE empfangen wurde.
b) beim Verbindungsaufbau vereinbart wurde, daß in der Datenphase ohne NEABX-Protokoll gearbeitet wird.

Die Struktur x_optd1 ist in der Datei neabx.h definiert.

```
  struct x_optd1 {
->          int x_optnr;            /* Optionsnummer */
<-          int x_code;             /* Nachrichtencode */
<-          int x_strukt;           /* Nachrichtenstruktur */
  };
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTD1.

x_code
bezeichnet den Nachrichtencode:

X_ASCII	Die eingegangenen Daten sind in ASCII codiert.
X_EBCDIC	Die eingegangenen Daten sind in EBCDIC codiert.
X_TRANS	Die eingegangenen Daten sind transparent.
X_UNDEF	NEABX hat keine Informationen über den Code. Die Daten liegen in dem Code vor, in dem sie der Partner gesendet hat.

Bei ISO-CCP- oder NEA-CCP-Anschluß liegt ein mitgeliefertes Benutzerdienstprotokoll in dem Code vor, in dem es der Partner gesendet hat. Die Benutzerdienstprotokolle sind somit transparent.

x_strukt
Nachrichtenstruktur. Folgende Werte sind möglich:

X_ETB	weiteres Gruppenelement der Teilgruppe folgt.
X_ETX	letztes oder einziges Gruppenelement einer Teilgruppe, weitere Teilgruppe folgt.
X_ETBEOT	letztes Gruppenelement einer Gruppe.
X_ETXEOT	letzte oder einzige Teilgruppe einer Gruppe.

Ergebnis

X_OK
Die Dateneinheit ist vollständig gelesen.

X_DATASTOP
Die Dateneinheit ist vollständig gelesen. Wenn Sie Daten senden wollen, müssen Sie das Ereignis X_DATAGO abwarten.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Hinweis

1. Eine mit x_datain() empfangene Dateneinheit kann kürzer oder länger als die mit x_datarq() gesendete Dateneinheit sein. Ist sie kürzer, dann steht im Indikator x_chain X_MORE und x_event() zeigt mit dem Ereignis X_DATAIN eine zum Empfang bereitstehende weitere Dateneinheit an.

2. Wenn Sie nicht bereit sind, Daten zu empfangen, können Sie mit dem Aufruf x_datastop() den Datenfluß stoppen. Damit verhindern Sie, daß NEABX der lokalen TS-Anwendung das Ereignis X_DATAIN zustellt. Eine einmal mit X_DATAIN angezeigte Dateneinheit müssen Sie jedoch immer vollständig abholen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>

#define         TRUE 1
#define         FALSE 0
#define         MAXLNG  16000
    .
    .
int     rc;                             /* Returncode */
int     datastop = FALSE;               /* Indikator Datenfluss */
int     debug = 2;                      /* mehr Information für Datenphase */
int     tref;                           /* Transportreferenz */
int     tidul;                          /* max. Länge einer Dateneinheit */

char    e_buf[MAXLNG];                  /* Empfangspuffer für Nachricht */
char    *e_bufpt;                       /* aktueller Pufferzeiger */
int     e_bufl;                         /* aktuelle Übertragungslänge */
int     e_bufcnt = 0;                   /* aktueller Übertragungszähler */
int     chain;                          /* Indikator für Nachrichtenende */

struct  x_opti1 option = { X_OPTI1 };   /* Struktur der x_info()-Option  */
    .
    .
e_bufpt = e_buf;
    .
    .
for (;;) {
    switch (x_event(&tref, X_WAIT, NULL)) {
    case X_DATAIN:
        if (x_info(&tref, &option) == X_ERROR) {
            fprintf(stderr, "Fehler 0x%x bei x_info\n", x_error());
            exit();
        }
        tidul = option.x_maxl;
        e_bufl = tidul;
```

```
        if((rc = x_datain(&tref, e_bufpt, &e_bufl, &chain, NULL)) == X_ERROR) {
            fprintf(stderr, "Fehler 0x%x bei x_datain für tref %x\n",
                x_error(), tref);
            exit();
        }
        if (rc == X_DATASTOP) datastop = TRUE;
        if (e_bufpt + e_bufl > &e_buf[MAXLNG]) {
            fprintf(stderr, "Puffer reicht nicht für Nachricht\n");
            exit();
        }
        e_bufpt += e_bufl;  e_bufcnt += e_bufl;
        if (chain == X_END) {
            if (debug > 1)
                fprintf(stderr, "Nachricht vollständig empfangen\n");
            break;
        }
        break;
        .
        .
    }
    .
    .
}
```

Der Prozeß wartet auf das nächste Ereignis. Wenn eine Partneranwendung Daten an die lokale TS-Anwendung sendet (Ereignis X_DATAIN), enthält tref die Transportreferenz der Verbindung, auf der die Dateneinheit empfangen wurde.
Die Länge des Datenbereichs wird gleich der Länge einer Dateneinheit für diese Verbindung gewählt. Dadurch ist sichergestellt, daß ein erfolgreicher Aufruf stets die ganze Dateneinheit übergibt. Wenn chain = X_END dem Prozeß mitteilt, daß die Nachricht vollständig empfangen wurde, wird die Verarbeitung fortgesetzt. Andernfalls wartet der Prozeß wieder auf das nächste Ereignis.

Daten senden - data request

Mit x_datarq() senden Sie eine Dateneinheit an die empfangende TS-Anwendung. Mit tref geben Sie an, auf welcher Verbindung Sie die Daten senden wollen. x_info() liefert Ihnen die maximale Länge einer Dateneinheit, die Sie auf dieser Verbindung senden können. Sie ist abhängig vom Transportsystem.

Ist die Nachricht, die Sie senden wollen, länger als eine Dateneinheit, so müssen Sie x_datarq() mehrmals hintereinander aufrufen. Mit dem Wert des Indikators x_chain teilen Sie NEABX mit, ob noch weitere Dateneinheiten, die zur Nachricht gehören, folgen.

Wenn x_datarq() als Ergebnis X_DATASTOP liefert, wurde die Dateneinheit übernommen, der Datenfluß aber für diese Verbindung gesperrt. Dies kann auf Initiative der empfangenden TS-Anwendung mit x_datastop() oder durch NEABX geschehen, wenn der lokale Zwischenspeicher überzulaufen droht. Sie müssen dann mit x_event() das Ereignis X_DATAGO abwarten, bevor Sie mit x_datarq() weiter auf dieser Verbindung senden können.

```
int x_datarq(tref,x_datap,x_datal,x_chain,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier geben Sie die Transportreferenz der Verbindung an, auf der Sie Daten senden wollen.

-> char *x_datap
Zeiger auf einen Bereich, in dem die Daten stehen, die Sie senden wollen. Ist x_opt ungleich NULL, muß dieser Bereich für später hinzugefügte Protokollheader um X_DRQPHL (Datarq-Protokoll Header Länge) größer angelegt werden, als es den effektiv zu sendenden Daten entspricht.
Die Benutzerverbindungsnachricht muß in diesem Bereich linksbündig stehen.

-> int *x_datal
Zeiger auf eine Längenangabe.

Fall x_opt = NULL (siehe x_opt):
*x_datal entspricht genau der Länge der zu sendenden Daten aus x_datap.
Maximale Angabe in *x_datal = x_maxl - X_DRQPHL ! *)
Maximum der transportierten Benutzerdaten pro x_datarq:
x_maxl - X_DRQPHL ! *)
Minimum von *x_datal (1Byte Sendedaten): *x_datal = 1.

Anmerkung
x_maxl ist die maximale Länge einer Dateneinheit, die auch später eingebrachte Protokollheader transportieren muß. Sie ist mit x_info zu ermitteln und wird auch TIDU-Länge genannt.

*) Das Maximum ist deshalb verkürzt, weil andere Instanzen ein Protokoll in der Länge X_DRQPHL anfügen.

Fall x_opt != NULL (siehe x_opt):
*x_datal muß um X_DRQPHL größer angegeben werden, als es den zu sendenden Daten entspricht.
Maximale Angabe in *x_datal = x_maxl.
Maximum von Benutzerdaten pro x_datarq:
x_maxl - X_DRQPHL.
Minimum von *x_datal (1Byte Sendedaten):
*x_datal = 1 + X_DRQPHL.

-> int *x_chain
Zeiger auf einen Indikator, mit dem Sie NEABX anzeigen, ob noch weitere Dateneinheiten zur Nachricht gehören.

Folgende Werte sind möglich:

X_MORE	Es folgen weitere Dateneinheiten der Nachricht, für jede Dateneinheit ist xadatarq() erneut aufzurufen.
X_END	Es sind keine weiteren Dateneinheiten vorhanden. Die Nachricht ist fertig übertragen.

-> union {struct x_optd1 optd1;} *x_opt
Zeiger auf eine Variante, die die Struktur x_optd1 oder die Angabe NULL enthält. Die Angabe NULL ist obligatorisch, wenn

a) eine weitere Dateneinheit einer Nachricht gesendet wird, also die vorhergehende Dateneinheit mit *x_chain = X_MORE gesendet wurde.
b) beim Verbindungsaufbau vereinbart wurde, daß in der Datenphase ohne NEABX-Protokoll gearbeitet wird.

Die Struktur x_optd1 ist in der Datei neabx.h definiert.

```
  struct x_optd1 {
->          int x_optnr;            /* Optionsnummer */
->          int x_code;             /* Nachrichtencode */
->          int x_strukt;           /* Nachrichtenstruktur */
  };
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTD1.

x_code
bezeichnet den Nachrichtencode der Daten in x_udatap:

X_ASCII Die zu sendenden Daten sind in ASCII codiert.

X_EBCDIC Die zu sendenden Daten sind in EBCDIC codiert.

X_TRANS Die zu sendenden Daten sind transparent.

Bei Stationskopplung ist die Angabe in x_code nicht relevant. Die Daten müssen in dem Code vorliegen, den der Partner erwartet.

Bei ISO-CCP- oder NEA-CCP-Anschluß muß ein mitgeliefertes Benutzerdienstprotokoll in dem Code vorliegen, den der Partner erwartet. Die Benutzerdienstprotokolle sind somit transparent.

x_strukt
Nachrichtenstruktur. Anzugeben ist X_ETX für Stationskopplung und X_ETXEOT für Rechnerkopplung.

Ergebnis

X_OK
Aufruf erfolgreich, Sie können weiter senden.

X_DATASTOP
Aufruf erfolgreich, Sie dürfen aber erst weiter senden, wenn das Ereignis X_DATAGO eingetroffen ist.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>

#define         MAXLNG  16000
    .
    .
int     debug = 2;                      /* mehr Information für Datenphase  */
int     rc;                             /* Returncode                       */
int     tref;                           /* Transportreferenz                */
int     chainmxl;                       /* Max.Lng.v.Ben.daten pro x_datarq */

char    s_buf[MAXLNG];                  /* Sendepuffer für Nachricht        */
int     msgsize = MAXLNG;               /* Länge der Nachricht              */
char    *s_bufpt;                       /* aktueller Pufferzeiger           */
int     s_bufl;                         /* aktuelle Übertragungslänge       */
int     s_bufcnt = 0;                   /* aktueller Übertragungszähler     */
int     chain;                          /* Indikator für Nachrichtenende    */

struct  x_opti1 option = { X_OPTI1 };   /* Struktur der x_info()-Option     */
    .
    .
s_bufpt = s_buf;
for (i = 0; i <= msgsize; i++)
    s_buf[i] = '*';
    .
    .
if (x_info(&tref, &option) == X_ERROR) {
    fprintf(stderr, "Fehler 0x%x bei x_info\n", x_error());
    exit();
}
chainmxl = option.x_maxl;           /* Für Ausgaben mit NEABX-Protokoll     */
                                    /* (d.h. in x_datarq x_opt != NULL)     */
    .                               /* muß an dieser Stelle                 */
    .                               /* chainmxl = option.x_maxl - X_DRQPHL; */
    .                               /* geschrieben werden.                  */
    .
    .
switch (x_event(&tref, X_CHECK, NULL)) {
case X_NOEVENT:
```

```
    while ((s_bufl = msgsize - s_bufcnt) > 0) {
        if (s_bufl > chainmxl) {
            chain = X_MORE;  s_bufl = chainmxl;
        } else
            chain = X_END;
        if ((rc = x_datarq(&tref, s_bufpt, &s_bufl, &chain, NULL)) == X_ERROR) {
            fprintf(stderr, "Fehler 0x%x bei x_datarq für tref %x\n",
                x_error(), tref);
            exit();
        }
        s_bufpt += s_bufl;  s_bufcnt += s_bufl;
        if (debug > 1 && chain == X_END)
            fprintf(stderr, "Nachricht vollständig gesendet\n");
        if (rc == X_DATASTOP)
            break;
    }
    break;
    .
    .
}
```

Der Prozeß prüft, ob für eine bestehende Verbindung ein Ereignis ansteht. Wenn nicht (Ergebnis X_NOEVENT), so enthält tref unverändert die Transportreferenz der bestehenden Verbindung, die Nachricht kann gesendet werden.
Wenn der zu sendende Teil der Nachricht länger als die transportierbaren Daten ist (chainmxl), wird als Übertragungslänge das oben beschriebene Maximum für *x_datal gewählt und der Indikator chain = X_MORE gesetzt. Andernfalls wird als Übertragungslänge gerade die Länge des zu sendenden Teils der Nachricht angegeben und chain = X_END teilt NEABX mit, daß keine weitere Dateneinheit folgt.
Wenn der Aufruf von x_datarq() das Ereignis X_DATASTOP meldet, kann keine weitere Dateneinheit gesendet werden.

Datenfluß stoppen - datastop

x_datastop() sperrt den Datenfluß auf der angegebenen Verbindung. Sie teilen NEABX mit, daß Sie nicht bereit sind, für diese Verbindung Daten zu empfangen. Ein bereits angezeigtes Ereignis X_DATAIN müssen Sie zuvor noch entgegennehmen. Der Aufruf bewirkt im einzelnen:

- Die lokale TS-Anwendung erhält das Ereignis X_DATAIN nicht mehr zugestellt. Sie können aber währenddessen andere NEABX-Funktionen aufrufen, z.B. eine weitere Verbindung aufbauen, über die Sie die ankommenden Daten weitergeben wollen.
- Die sendende TS-Anwendung erhält beim Aufruf x_datarq() für diese Verbindung das Ergebnis X_DATASTOP. Sie darf dann keine Daten mehr senden.

Erneut freigeben können Sie den Datenfluß mit x_datago().

```
int x_datastop(tref)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, für die Sie den Datenfluß stoppen wollen.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Hinweis

Wenn Sie bei x_event() das Ereignis X_DATAIN erhalten haben, müssen Sie die anstehende Dateneinheit vollständig lesen. x_datastop() bewirkt nur, daß kein weiteres Ereignis X_DATAIN zugestellt wird.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
    .
    .
int   tref;       /* Transportreferenz */
    .
    .
if (x_datastop(&tref) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_datastop für tref %x\n",
        x_error(), tref);
```

Der Aufruf stoppt den Datenfluß für die mit tref angegebene Verbindung.

Abmelden bei NEABX - detach process

x_detach() meldet den laufenden Prozeß für die angegebene TS-Anwendung bei NEABX ab. Falls noch Verbindungen dieses Prozesses bestehen, baut sie NEABX ab. Sobald Sie den letzten Prozeß einer TS-Anwendung abgemeldet haben, ist die TS-Anwendung nicht mehr bekannt. Verbindungsanforderungen für diese TS-Anwendung werden dann nicht mehr angenommen.

```
int x_detach(name, NULL)
```

Parameter

-> union x_address *name
Zeiger auf eine Variante, in der der LOKALE NAME der TS-Anwendung anzugeben ist. Der LOKALE NAME ist in der Datei neabx.h als Struktur x_myname definiert. Die Versorgung der Struktur mit den auf das jeweilige CCP abgestimmten Angaben erfolgt wie unter x_attach() beschrieben. Es wird derselbe LOKALE NAME angegeben wie beim Aufruf x_attach().

-> NULL
müssen Sie angeben, da es zur Zeit keine Optionen gibt.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
  .
  .
char    myname[] = "MX2 III-15";                    /* Anwendername */

struct x_opta1 opta1 = { X_OPTA1, X_ACTIVE, 1}; /* Prozeßparameter */
```

```
struct x_myname myaddress;                          /* LOKALER NAME */
    .
    .
if (x_attach(&myaddress, &opta1) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_attach\n", x_error());
    .
    .
if (x_detach(&myaddress, NULL) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_detach\n", x_error());
```

Die erforderlichen Vereinbarungen und Aufrufe für myaddress erfolgen wie im Beispiel zu x_attach().
Der Aufruf x_detach() meldet den Prozeß, der sich unter dem lokal eindeutigen Namen "MX2 III-15" bei NEABX angemeldet hatte, wieder bei NEABX ab.

Verbindungsabbau entgegennehmen - disconnection indication

x_disin() rufen Sie auf, wenn Sie das Ereignis X_DISIN erhalten haben. Wenn Sie x_disin() nicht aufrufen, baut NEABX dennoch die Verbindung ab. Mit x_disin() erhalten Sie Auskunft darüber, ob NEABX oder die ferne TS-Anwendung die Verbindung abgebaut haben. In diesem Fall übergibt NEABX beim Aufruf x_disin() die Benutzerdaten, die die ferne TS-Anwendung beim Verbindungsabbau mitgeschickt hat, sofern das verwendete Transportsystem diese Möglichkeit bietet.

```
int x_disin(tref,reason,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz ein, die Sie erhalten, wenn der Aufruf von x_event() das Ereignis X_DISIN meldet.

<- int *reason
Zeiger auf die Ursache des Verbindungsabbaus. Folgende Werte sind möglich:

X_USER Die ferne TS-Anwendung hat die Verbindung abgebaut.

sonst NEABX oder das Transportsystem haben die Verbindung abgebaut.
Die möglichen Werte finden Sie in der Datei neabx.h.

<- union {struct x_optc2 optc2;} *x_opt
Zeiger auf eine Variante, in die NEABX eine Struktur x_optc2 einträgt. Mit dieser Struktur können Sie die Information abfragen, die die ferne TS-Anwendung beim Verbindungsabbau mitgeliefert hat.
Geben Sie statt des Zeigers NULL an, wirft NEABX diese Informationen weg.
Die Struktur x_optc2 ist in der Datei neabx.h definiert.

```
  struct x_optc2 {
->          int x_optnr;            /* Options-Nr. */
<-          char *x_udatap;         /* Datenpuffer */
<>          int x_udatal;           /* Länge des Datenp. */
  } ;
```

x_optnr
: Optionsnummer. Anzugeben ist X_OPTC2.

x_udatap
: Zeiger auf einen Datenbereich. In diesen Bereich trägt NEABX die Benutzerdaten ein, die die ferne TS-Anwendung beim Verbindungsabbau mitgeschickt hat.
Den Bereich müssen Sie so groß wählen, daß die empfangene Nachricht hineinpaßt. Die maximal zulässige Länge ist vom verwendeten Transportsystem abhängig und maximal X_MSG_SIZE Byte groß.

x_udatal
: Vor dem Aufruf geben Sie die Länge des bereitgestellten Datenbereichs an, mindestens aber X_MSG_SIZE. Beim Aufruf trägt NEABX die Länge der empfangenen Nachricht ein.

Ergebnis

X_OK
: Der Aufruf war erfolgreich.

X_ERROR
: Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
  .
  .
int     debug = 1;                      /* mehr Information */
int     tref;                           /* Transportreferenz */
int     reason;                         /* Grund für Verbindungsabbau */
  .
  .
switch (x_event(&tref, X_WAIT, NULL)) {
case X_DISIN:
    if (x_disin()(&tref, &reason, NULL) == X_ERROR) {
```

```
        fprintf(stderr, "Fehler 0x%x bei x_disin() tref %x\n",
            x_error(), tref);
        exit (0);
    }
    if (debug)
        fprintf(stderr, "Verbindungsabbau erhalten, tref %x, reason %d\n",
            tref, reason);
    break;
    .
    .
}
```

Der Prozeß wartet auf das nächste Ereignis. Wenn eine ferne TS-Anwendung eine Verbindungsanforderung ablehnt oder eine bestehende Verbindung abbaut (Ereignis X_DISIN), so enthält tref die Transportreferenz der abgebauten bzw. abgelehnten Verbindung. Der Aufruf x_disin() liefert die Ursache des Verbindungsabbaus.

Verbindung abbauen - disconnection request

Mit x_disrq() können Sie:

- eine bestehende Verbindung abbauen oder
- die Verbindungsanforderung einer fernen TS-Anwendung ablehnen.

NEABX stellt der fernen TS-Anwendung in beiden Fällen das Ereignis X_DISIN und die Ursache (reason) zu.
Jede TS-Anwendung kann die Verbindung abbauen, unabhängig davon, welche die Verbindung aktiv aufgebaut hat. Wenn der Aufruf x_disrq() erfolgreich war, ist die Verbindung abgebaut. Auch NEABX kann Verbindungen abbauen, wenn NEABX-interne Gründe es erfordern.

Der Aufruf x_disrq() kann Dateneinheiten überholen, die noch unterwegs sind. Diese gehen dann verloren. Um das zu verhindern, können Sie z.B. logische Quittungen vereinbaren und x_disrq() erst aufrufen, wenn Sie die positive Quittung für die letze Dateneinheit erhalten haben.

BCAM bietet keine Schnittstelle, Benutzerdaten der Anwendung zu übergeben.

```
int x_disrq(tref,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, die Sie abbauen wollen. Wenn Sie eine Verbindungsanforderung ablehnen wollen, die beim von x_event() mit dem Ereignis X_CONIN angezeigt wird, so liefert x_event() auch die Transportreferenz der abzuweisenden Verbindung.

-> union {struct x_optc2 optc2;} *x_opt
Zeiger auf eine Variante, für den NULL anzugeben ist.
Zu heutigen Anwendungen im BS2000 über BCAM können keine Benutzerdaten übergeben werden.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Hinweis

Beendet sich ein Prozeß, bevor die bestehenden Verbindungen abgebaut wurden, so baut NEABX die Verbindungen ab.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
  .
  .
char    partname[] = "MX2 IV-1";                    /* Anwendername Partner */

struct x_myname    myaddress;                       /* LOKALER NAME */
union  x_partaddr  partaddr;                        /* TRANSPORTADRESSE */
  .
  .
if (x_conrq(&tref, &partaddr, &myaddress, NULL) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_conrq für tref %x\n",
        x_error(), tref);
  .
  .
if (x_disrq(&tref, NULL) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_disrq für tref %x\n",
        x_error(), tref);
```

Die erforderlichen Vereinbarungen und Aufrufe für myaddress und partaddr erfolgen wie im Beispiel zu x_attach() und x_conrq().
Der Aufruf x_disrq() baut die Verbindung der TS-Anwendung "MX2 IV-1" ab. In tref ist dabei die Transportreferenz angegeben, die NEABX beim Aufruf x_conrq() für diese Verbindung eingetragen hat.

Fehlercodes abfragen - error

x_error() rufen Sie auf, wenn ein anderer NEABX-Aufruf das Ergebnis X_ERROR hatte. x_error() liefert seinerseits als Ergebnis einen Fehlercode.

```
int x_error()
```

Parameter: keine

Ergebnis

X_NOERROR
Kein Fehler.

X_UNSPECIFIED
Nicht näher spezifizierbarer Fehler.

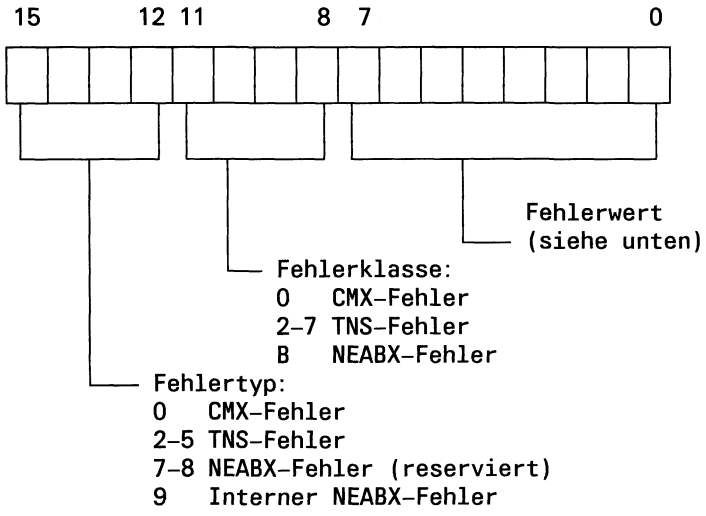
Diese beiden Fehlertypen sind systemunspezifisch.

sonst. Systemspezifischer Fehlertyp. Der folgende Überblick gibt dazu weitere Hinweise.

Überblick

Die Ausführungen beziehen sich auf CMX V2.1A. Die möglichen Meldungen zu den Aufrufen an ICMX(NEA) entstehen entweder in der NEABX-Bibliothek im Benutzerprozeß oder im Betriebssystemkern. Die Meldungen des Betriebssystemkerns können noch danach unterschieden werden, ob sie in NEABX oder im CMX selbst erzeugt werden oder aus Betriebssystemaufrufen resultieren.

Die Fehlermeldung wird in 16 Bit dargestellt:



Die Fehlermeldung ist von links (Bit 15) her auszuwerten. Falls das höherwertige Byte hexadezimal "9B" (= interner NEABX-Fehler) ist, sind die Fehlerwerte hier im folgenden aufgeführt, alle anderen Fehlerwerte sind im Abschnitt 6.1 beim Funktionsaufruf t_error bzw. eingangs des Abschnitts 6.2 beschrieben.

Folgende von NEABX erzeugte Fehlerwerte sind definiert:

0	X_NOERROR	Kein Fehler
1	X_DIDSSSTK	Partner hat weder ETB noch ETX bei DSS-Anschluß angeben
2	X_NOTNEABX	Nicht im X_NEABX Modus
3	X_MAXDAT	Beim Verbindungsaufbau mehr als X_MSGF_SIZE Benutzerdaten erhalten
4	X_BADLEN	Ungültige Datenpufferlänge
5	X_BADTRANS	Ungültiges Transportsystem
6	X_BADKOPP	Ungültige Kopplungsart
7	X_BADTABLE	Kein Tabelleneintrag vorhanden
8	X_BADPROT	Falsches Protokoll-Identifikations-Byte erhalten
9	X_DIDSSCDE	Bei DSS-Anschluß Daten nicht in EBCDI-Code
10	X_SENDQUIT	Fehler beim Senden eines QUITTUNG-Protokollelements
11	X_XSNDQUIT	Fehler beim Senden eines QUITTUNG-Protokollelements für Vorrangdaten
12	X_NOOPTDSS	Kein x_opt bei Datenstationskopplung angegeben
13	X_BADXCODE	Falscher Transmission-Code angegeben oder erhalten
14	X_BADPRPI	Unbekanntes Protokoll-Identifikations-Byte erhalten
15	X_NOTCNPE	CONNECT-Protokollelement erwartet, aber nicht erhalten
16	X_NOTCNATT	CONNECT-ATTENTION-Protokollelement erwartet, aber nicht erhalten
17	X_NOTDTPE	DATA-Protokollelement erwartet, aber nicht erhalten
18	X_QUITPE	x_datain hat ein QUITTUNG-Protokollelement erhalten

19	X_DATAPE	x_datain hat DATA-Protokollelement erhalten, aber nicht erwartet	
20	X_BADPVBYTE	Fehlerhaftes Protokoll-Versions-Byte erhalten	
21	X_NONEBYTE	Bei DSS-Anschluß kein NEABX in der Datenphase vereinbart	
22	X_BADDTPELI	Falsche Länge im Längen-Indikator-Byte erhalten	
23	X_BADSTRUKT	Falsche Struktur-Angabe in x_optd1	
24	X_DS_RDLF	Fehler beim TNS-Aufruf	
25	X_BADREDIR	Ungültige Verbindungsumlenkung	
42	X_BADMSGLEN	Ungültige Benutzernachrichtenlänge	
43	X_NOXDRDSS	DSS darf x_xdatrq nicht aufrufen	
44	X_NOXDIDSS	DSS darf x_datin nicht aufrufen	
45	X_WREPBIT	Falscher Eintrag im Repeat-Puffer	
46	X_NOINFO	Keine TIDU-Länge bestimmbar	
47	X_BADXREAD	t_xdatin liefert Wert größer Null	
48	X_QOVERFLOW	Transportquittung kann nicht mehr gespeichert werden	
49	X_NOOPT	Kein x_opt Pointer angegeben	
50	X_WPARAMETER	Fehlerhafter Parameter, z.B. falsche x_optnr angegeben	
51	X_NVERR1	unzulässige Eingabeparam. (x_neavi, x_neavo): Nullangaben in x_udatal, x_udatap, *x_udatal zu kurz: x_neavo: Reserve für NEABX-Header (8By) nicht berücksichtigt oder bei Rk. OPCH-Länge=1 nicht berücksichtigt, u.a.	siehe auch neabx.h im Anhang
52	X_NVERR2	unzulässige Optionsnummer (x_neavi, x_neavo)	
53	X_NVERR3	unzulässige Längen in NEABV-Nachricht (x_neavi/o)	
54	X_NVERR4	unzulässige Ang. zu x_init (x_neavo)	
55	X_NVERR5	unzulässige Ang. zu x_opch/x_bvmsg/x_npw (x_neavo)	
58	X_FOPENERR	Fehler beim Datei-Eröffnen: fopen nea.trc.<pid>	
59	X_WXOPT	Falsche x_opt Angabe; z.B. x_opt != NULL; x_opt != NULL, obwohl kein NEABX-Protokoll; x_opt != NULL, obwohl zweite und folgende Dateneinheiten bei x_datarq und x_datain; ...	

Beispiel

Bei den Beispielen zu den anderen NEABX-Aufrufen sind x_error()-Aufrufe enthalten.

Ereignis abwarten oder abfragen - event

x_event() meldet, ob ein Ereignis ansteht und die Art des Ereignisses. Abhängig vom Parameter x_cmode

- wartet x_event() solange, bis ein Ereignis ansteht oder
- prüft x_event(), ob ein Ereignis ansteht und meldet X_NOEVENT, wenn das nicht der Fall ist.

Liegen mehrere Ereignisse vor, werden sie der Reihe nach angezeigt. Das Ereignis X_XDATIN kann jedoch Dateneinheiten der gleichen Verbindung überholen. Auch das Ereignis X_DISIN kann Dateneiheiten überholen und zerstört sie dadurch.

```
int x_event(tref,x_cmode,NULL)
```

Parameter

<- int *tref
Zeiger auf die Transportreferenz. Hier trägt NEABX die Transportreferenz der Verbindung ein, zu der das gemeldete Ereignis gehört. Bei X_NOEVENT oder X_ERROR ist der Inhalt von tref nicht definiert.

-> int x_cmode
gibt an, ob x_event() auf ein Ereignis warten soll oder nicht.

X_WAIT
x_event() wartet auf das nächste Ereignis (synchrone Verarbeitung). Wenn Sie x_event() mit X_WAIT aufrufen, schläft der Prozeß, bis ein Ereignis ansteht.

Mit einem Signal läßt sich der Aufruf abbrechen. Man kann z.B. die Funktion alarm() verwenden, um x_event() zeitlich zu überwachen. Erlaubt sind alle Signale außer SIGTERM. Wird x_event() durch ein Signal unterbrochen, ist das Ergebnis X_NOEVENT.

X_CHECK
x_event() prüft, ob ein Ereignis ansteht (asynchrone Verarbeitung). Wenn nicht, dann kehrt x_event() mit X_NOEVENT zurück.

-> NULL
müssen Sie angeben, da es noch keine Optionen gibt.

Ergebnis

X_NOEVENT
falls x_cmode = X_CHECK: kein Ereignis da.
falls x_cmode = X_WAIT : Abbruch, z.B. durch ein Signal

Der Inhalt von tref ist nicht definiert.

X_DATAIN
Daten stehen zum Empfangen bereit.
Von NEABX erwartete Reaktion: x_datain().

Hinweis
NEABX zeigt dieses Ereignis nicht an, solange der Datenfluß gestoppt ist, das heißt, wenn der empfangende Prozeß für diese Verbindung x_datastop() gegeben hat.

X_DATAGO
Die ferne TS-Anwendung hat den Datenfluß mit x_datago() freigegeben. Die lokale TS-Anwendung darf wieder Daten senden.
Mögliche Reaktion: x_datarq().

Hinweis
Wenn für diese Verbindung die Benutzung von Vorrangdaten erlaubt ist, können auch wieder Vorrangdaten gesendet werden.

X_XDATIN
Vorrangdaten stehen zum Empfangen bereit.
Von NEABX erwartete Reaktion: x_xdatin().

Hinweis
NEABX zeigt dieses Ereignis nur an, wenn beim Verbindungsaufbau die Benutzung von Vorrangdaten vereinbart wurde.
Solange der Vorrangdatenfluß gestoppt ist, der emfangende Prozeß also für diese Verbindung x_xdatstop() gegeben hat, wird dieses Ereignis nicht angezeigt.

X_XDATGO
Die ferne TS-Anwendung hat den Vorrangdatenfluß mit x_xdatgo() freigegeben. Die lokale TS-Anwendung darf wieder Vorrangdaten senden.
Mögliche Reaktion: x_xdatrq().

Hinweis

NEABX zeigt dieses Ereignis nur an, wenn beim Verbindungsaufbau die Benutzung von Vorrangdaten vereinbart wurde.

X_CONIN

Eine Partneranwendung möchte eine Verbindung zur lokalen TS-Anwendung aufbauen (ankommender Ruf). Sie müssen diese Verbindungsanforderung mit x_conin() entgegennehmen und dann mit x_conrs() bestätigen oder mit x_disrq() ablehnen.
Von NEABX erwartete Reaktion: x_conin(), dann x_conrs() oder x_disrq().

X_CONCF

Die ferne TS-Anwendung hat die Verbindungsanforderung mit x_conrs() angenommen. Sie müssen diese Antwort mit x_concf() entgegennehmen. Dann steht die Verbindung.
Von NEABX erwartete Reaktion: x_concf().

X_DISIN

Entweder hat die gerufene TS-Anwendung eine Verbindungsanforderung abgelehnt oder die ferne TS-Anwendung oder NEABX haben eine bestehende Verbindung abgebaut. Sie müssen diese Anzeige mit x_disin() entgegennehmen.
Von NEABX erwartete Reaktion: x_disin().

X_REDIN

Ein anderer Prozeß der TS-Anwendung möchte eine Verbindung auf diesen Prozeß umlenken. Sie müssen die Verbindung mit x_redin() annehmen.
Von NEABX erwartete Reaktion: x_redin().

X_REPCCF

NEABX hat die Verbindung noch nicht gänzlich herstellen können. Daher muß der Aufruf x_concf() wiederholt werden.
Von NEABX erwartete Reaktion: x_concf(), siehe dort.

X_REPCIN

NEABX hat die Verbindungsanforderung noch nicht ganz entgegennehmen können. Daher muß der Aufruf x_conin() wiederholt werden.
Von NEABX erwartete Reaktion: x_conin(), siehe dort.

X_REPCRQ
NEABX hat die Verbindung angefordert, die Anforderung wurde nicht vollständig von der fernen TS-Anwendung entgegengenommen. Daher muß der Aufruf x_conrq() wiederholt werden.
Von NEABX erwartete Reaktion: x_conrq(), siehe dort.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Der Inhalt von tref ist nicht definiert.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <stdio.h>
    .
    .
int   tref;       /* Transportreferenz */
int   mode;       /* Modus für x_event() */
int   event;      /* Ereignis */
    .
    .
mode = X_WAIT;
for (;;) {
    switch (event = x_event(&tref, mode, NULL)) {
    case X_CONCF:
                  .....
            break;
    case X_NOEVENT:
                  .....
            mode = X_CHECK;
            break;
      .
      .
    case X_ERROR:
            fprintf(stderr, "Fehler 0x%x bei x_event\n", x_error());
            mode = X_CHECK;
            break;
    default:
            fprintf(stderr, "Unerwarteter event %d\n", event);
            break;
    }
}
```

Das Beispiel zeigt, wie x_event() in einer Schleife aufgerufen wird. Alle Reaktionen auf die möglichen Ereignisse sind in den jeweiligen case-Blöcken zusammengefaßt.

Informationen über NEABX-Konstante - information

x_info() liefert die maximal mögliche Länge einer Dateneinheit für die angegebene Verbindung. Diese Größe hängt vom verwendeten Transportsystem ab. Sie brauchen sie für die Aufrufe zum Datentransport.

```
int x_info(tref,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier geben Sie die Transportreferenz der Verbindung ein, zu der Sie die maximal mögliche Länge einer Dateneinheit wissen wollen.

-> union {struct x_opti1 opti1;} *x_opt
Zeiger auf eine Variante, in die NEABX eine Struktur x_opti1 einträgt. Die Struktur x_opti1 ist in der Datei neabx.h definiert.

```
  struct x_opti1 {
->        int x_optnr;          /* Optionsnummer */
<-        int x_maxl;           /* Länge einer TIDU */
  };
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTI1.

x_maxl
Hier trägt NEABX die Länge einer zu übertragenden Dateneinheit ein. Dieser Wert gibt an, wieviel Byte eine Nachricht haben darf, um sie auf dieser Verbindung mit einem Aufruf x_datarq() senden bzw. mit x_datain() empfangen können.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
    .
    .
int     tref;                           /* Transportreferenz */
int     tidul;                          /* Länge einer Dateneinheit */
struct  x_opti1 option = { X_OPTI1 };   /* Struktur der x_info()-Option  */
    .
    .
if (x_info(&tref,&option) == X_ERROR)
        fprintf(stderr, "Fehler 0x%x bei x_info\n", x_error());
tidul = option.x_maxl;
```

x_info()
liefert die Länge einer Dateneinheit, deren Wert der Variablen tidul zugewiesen wird.

Aufrufe des NEABV-Service

```
int x_neavi(x_udatap,x_udatal,x_opt)
```

```
int x_neavo(x_udatap,x_udatal,x_opt)
```

Die Erklärung der Aufrufe finden Sie im Abschnitt 7.

Umgelenkte Verbindung annehmen - redirection indication

Mit x_redin() nimmt ein Prozeß eine Verbindung an, die ein anderer Prozeß derselben TS-Anwendung auf ihn umgelenkt hat. Der Aufruf ist erforderlich, wenn x_event() das Ereignis X_REDIN anzeigt.

Beim Ereignis X_REDIN müssen Sie die umgelenkte Verbindung annehmen. Wenn Sie die Verbindung ablehnen wollen, können Sie sie nur mit x_redrq() weitergeben oder zum ursprünglichen Prozeß zurückgeben oder mit x_disrq() abbauen.

Mit dem Aufruf können Sie auch Benutzerdaten erhalten, die der umlenkende Prozeß mitgeschickt hat.

```
int x_redin(tref,pid,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportrefenz ein, die Sie beim Aufruf x_event() mit Ergebnis X_REDIN erhalten haben.

<- int *pid
Zeiger auf die Prozeßnummer. NEABX trägt hier die Nummer des Prozesses ein, der die Verbindung umlenkt.

<- union {struct x_optc2 optc2;} *x_opt
Zeiger auf eine Variante, in die NEABX eine Struktur x_optc2 einträgt. Mit dieser Struktur können Sie die Information abfragen, die der umlenkende Prozeß beim Aufruf x_redrq() mitgeliefert hat.
Die Struktur x_optc2 ist in der Datei neabx.h definiert.

```
  struct x_optc2 {
->          int x_optnr;            /* Options-Nr. */
<-          char *x_udatap;         /* Datenpuffer */
<>          int x_udatal;           /* Länge des Datenp. */
  } ;
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTC2.

x_udatap
Zeiger auf einen Bereich mit Benutzerdaten, die NEABX dem annehmenden Prozeß übergibt.
Den Bereich müssen Sie so groß wählen, daß die empfangene Nachricht hineinpaßt. Die empfangene Nachricht (Benutzerdaten, Platzreservierung) ist maximal X_RED_SIZE Byte lang.

x_udatal
Vor dem Aufruf geben Sie die Länge des bereitgestellten Datenbereichs an, minimal aber X_RED_SIZE. Beim Aufruf trägt NEABX die Länge der empfangenen Nachricht ein.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
    .
    .
int     tref;                           /* Transportreferenz */
int     pid;                            /* pid des umlenkenden Prozesses */
    .
    .
char    udatap[X_RED_SIZE];             /* Datenpuffer */
struct  x_optc2 optc2 = { X_OPTC2,      /* Options-Nr. */
                          &udatap[0], /* Datenpuffer */
                          X_RED_SIZE }/* Länge des Datenpuffers */
switch (x_event(&tref, X_WAIT, NULL)) {
case X_REDIN:
    if (x_redin(&tref, &pid, &optc2) == X_ERROR) {
        fprintf(stderr, "Fehler 0x%x bei x_redin(), tref %x, pid %d\n",
            x_error(), tref, pid);
        exit (1);
    }
    break;
    .
    .
}
```

Der Prozeß wartet auf das nächste Ereignis. Wenn ein anderer Prozeß der TS-Anwendung eine Verbindung auf diesen Prozeß umlenkt (Ereignis X_REDIN), enthält tref die Transportreferenz dieser Verbindung. Um die Prozeßnummer pidpar des umlenkenden Prozesses zu erhalten, wird x_redin() mit dieser Transportreferenz aufgerufen.

Verbindung umlenken - redirection request

x_redrq() gibt eine bestehende Verbindung an einen anderen Prozeß derselben TS-Anwendung weiter. Die Verbindung ist anschließend dem umlenkenden Prozeß nicht mehr bekannt. NEABX stellt dem gerufenen Prozeß das Ereignis X_REDIN zu.

Sie dürfen eine Verbindung nicht umlenken,

- wenn X_DATASTOP oder X_XDATSTOP vorliegt oder
- wenn der vorangegangene Aufruf von x_event() das Ergebnis X_NOEVENT ergeben hat.

Mit der Verbindungsumlenkung können Sie dem annehmenden Prozeß Benutzerdaten mitschicken.

```
int x_redrq(tref,pid,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Sie tragen hier die Transportreferenz der Verbindung ein, die Sie umlenken wollen.

-> int *pid
Zeiger auf die Prozeßnummer des gerufenen Prozesses. Hier geben Sie die Nummer des Prozesses an, zu dem Sie die Verbindung umlenken wollen.

-> union {struct x_optc2 optc2;} *x_opt
Zeiger auf eine Variante, in der eine Struktur x_optc2 anzugeben ist. Bei falsch versorgten Parametern ist das Ergebnis X_ERROR (x_error() liefert den Fehlercode X_WPARAMETER).
Mit dieser Struktur können Sie beim Verbindungsumlenken dem gerufenen Prozeß Benutzerdaten mitschicken, die er beim Aufruf x_redin() erhält.
Die Struktur x_optc2 ist in der Datei neabx.h definiert.

```
  struct x_optc2 {
->          int x_optnr;            /* Options-Nr. */
->          char *x_udatap;         /* Datenpuffer */
->          int x_udatal;           /* Länge des Datenp. */
  };
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTC2.

x_udatap
Zeiger auf einen Bereich mit Benutzerdaten, die NEABX dem annehmenden Prozeß übergeben soll, plus Platzreservierung in der Länge von X_RED_PL (z.Zt. 5 Byte) für das interne x_red-Protokoll.
Die Benutzerverbindungsnachricht muß in diesem Bereich linksbündig stehen.

x_udatal
Länge der Nachricht aus x_udatap plus X_RED_PL.
Maximalwert: X_RED_SIZE.
Maximum der transportierten Benutzerdaten pro x_redrq():
X_RED_SIZE minus X_RED_PL.
Minimale Angabe: X_RED_PL.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_IMPOSSIBLE
Die Umlenkungsanfrage wird z.Zt. nicht ausgeführt (interne Gründe).

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
    .
    .
int     tref;                       /* Transportreferenz */
int     pidson;                     /* pid des Sohnes nach fork() */
char    updatap []="<Benutzerdaten>"; /* Datenpuffer */
struct x_optc2 optc2 = { X_OPTC",   /* Options-Nr. */
                         &udatap[0], /* Datenpuffer */
```

```
                        size of      /* Länge des Datenpuffers */
                        (updatap) };
    .
    .
if ((pidson = fork()) > 0 ) {
        .
        .
    switch (x_event(&tref, X_WAIT, NULL)) {
    case X_CONIN:
            .
            .
        if (x_redrq(&tref, &pidson, &optc2) == X_ERROR) {
            fprintf(stderr, "Fehler 0x%x bei x_redrq für tref %x, pid %d\n",
                x_error(), tref, pidson);
            exit ();
        }
        break;
        .
        .
    }
}
```

Der Prozeß wartet auf das nächste Ereignis. Wenn eine ferne TS-Anwendung eine Verbindung zur lokalen TS-Anwendung anfordert, enthält tref die Transportreferenz. Der Vaterprozeß lenkt diese Verbindung auf einen Prozeß der gleichen TS-Anwendung um. Der Sohnprozeß hat die Prozeßnummer pidson. In diesem Beispiel sind die Prozesse verwandt, das ist aber nicht zwingend.

Vorrangdatenfluß freigeben - expedited data go

x_xdatgo() gibt den gesperrten Vorrangdatenfluß auf der angegebenen Verbindung frei. Der laufende Prozeß teilt NEABX mit, daß er wieder bereit ist, Vorrangdaten zu empfangen. Wenn die Benutzung von Vorrangdaten vereinbart ist, wird auch der Vorrangdatenfluß freigegeben, falls er gesperrt war. Dieser Aufruf ist nur bei Rechnerkopplung sinnvoll.

Der Aufruf bewirkt im einzelnen:

- Die lokale TS-Anwendung erhält das Ereignis X_XDATIN zugestellt, falls es ansteht.
- Die ferne TS-Anwendung erhält das Ereignis X_XDATGO zugestellt. Sie darf wieder Vorrangdaten senden.

```
int x_xdatgo(tref)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, für die Sie den Vorrangdatenfluß freigeben wollen.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beipiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
    .
    .
int   tref;       /* Transportreferenz */
    .
    .
if (x_xdatgo(&tref) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_xdatgo für tref %x\n",
        x_error(), tref);
```

Der Aufruf gibt den Vorrangdatenfluß für die mit tref angegebene Verbindung frei.

Vorrangdaten empfangen - expedited data indication

Mit x_xdatin() nimmt der laufende Prozeß seine von der fernen TS-Anwendung gesendeten Vorrangdaten entgegen. Zuvor muß NEABX bei x_event() das Ereignis X_XDATIN gemeldet haben. Dabei übergibt NEABX in tref die Transportreferenz der Verbindung, auf der die Vorrangdaten eingegangen sind. Die maximale Länge der Vorrangdaten ist abhängig vom Transportsystem und kann nie länger als X_EXP_SIZE Byte sein.

```
int x_xdatin(tref,x_datap,x_datal,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz ein, die Sie erhalten, wenn der Aufruf von x_event() das Ereignis X_XDATIN meldet.

<- char *x_datap
Zeiger auf einen Bereich, in den NEABX die empfangenen Vorrangdaten einträgt.
Ist x_opt gleich NULL, so übergibt NEABX die empfangenen Vorrangdaten der lokalen TS-Anwendung.
Ist x_opt ungleich NULL, so wird das NEABX-Protokoll von NEABX vor der Übergabe an die lokale TS-Anwendung entfernt. Ist kein NEABX-Protokoll vorhanden, ist das Ergebnis X_ERROR.

<- int *x_datal
Vor dem Aufruf geben Sie die Länge des Vorrangdatenbereichs x_datap an, mindestens aber X_EXP_SIZE.
Beim Aufruf trägt NEABX die Anzahl der eingetragenen Byte ein, die an die lokale TS-Anwendung übergeben werden.

<- union {struct x_optd1 optd1;} *x_opt
Zeiger auf eine Variante, die die Struktur x_optd1 oder die Angabe NULL enthält. Die Angabe NULL ist obligatorisch, wenn beim Verbindungsaufbau vereinbart wurde, daß in der Datenphase ohne NEABX-Protokoll gearbeitet wird.
Die Struktur x_optd1 ist in der Datei neabx.h definiert.

```
  struct x_optd1 {
->          int x_optnr;            /* Optionsnummer */
<-          int x_code;             /* Nachrichtencode */
<-          int x_strukt;           /* Nachrichtenstruktur */
  };
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTD1.

x_code
bezeichnet den Nachrichtencode. Angegeben ist X_TRANS, die empfangenen Vorrangdaten sind transparent.

x_strukt
Nachrichtenstruktur. Angegeben ist X_ETX. Vorrangdaten können nur ungeblockt empfangen werden.

Ergebnis

X_OK
Die Vorrangdaten wurden vollständig gelesen.

X_DATASTOP
Die Vorrangdaten wurden vollständig gelesen. Wenn Sie Daten senden wollen, müssen Sie das Ereignis X_DATAGO abwarten.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Hinweis

Wenn Sie nicht bereit sind, Vorrangdaten zu empfangen, können Sie mit x_xdatstop() den Vorrangdatenfluß stoppen. Damit verhindern Sie, daß NEABX der lokalen Anwendung das Ereignis X_XDATIN zustellt. Einmal mit X_DATAIN angezeigte Vorrangdaten müssen Sie jedoch immer vollständig abholen.

Beispiel

Siehe Beispiel für x_datain().

Vorrangdaten senden - expedited data request

Mit x_xdatrq() senden Sie Vorrangdaten an die empfangende TS-Anwendung, sofern beim Aufbau dieser Verbindung die Benutzung von Vorrangdaten vereinbart wurde. Mit tref geben Sie an, auf welcher Verbindung Sie die Vorrangdaten senden wollen. Die maximale Länge der Vorrangdaten ist abhängig vom Transportsystem und kann nie länger als X_EXP_SIZE Byte sein.

Dieser Aufruf ist nur bei Rechnerkopplung sinnvoll.

Vorrangdaten unterliegen einer eigenen Flußregelung, sie können normale Dateneinheiten überholen. Umgekehrt garantiert das Transportsystem, daß Vorrangdaten nie von normalen Dateneinheiten überholt werden.

Wenn x_xdatrq() als Ergebnis X_XDATSTOP liefert, wurden die Vorrangdaten übernommen, der Vorrangdatenfluß aber für diese Verbindung gesperrt. Dies kann auf Initiative der empfangenden Partneranwendung mit x_xdatstop() oder durch NEABX geschehen, wenn der lokale Zwischenspeicher überzulaufen droht. Sie müssen dann mit x_event() das Ereignis X_XDATGO oder X_DATAGO abwarten, bevor Sie weitere Vorrangdaten auf dieser Verbindung senden können.

```
int x_xdatrq(tref,x_datap,x_datal,x_opt)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier geben Sie die Transportreferenz der Verbindung an, auf der Sie Vorrangdaten senden wollen.

-> char *x_datap
Zeiger auf einen Bereich, in dem die Daten stehen, die Sie senden wollen. Ist x_opt ungleich NULL, muß dieser Bereich für später hinzugefügte Protokollheader um X_DRQPHL (Datarq-Protokoll Header Länge) größer angelegt werden, als es den effektiv zu sendenden Daten entspricht. Er ist aber niemals größer als X_EXP_SIZE.
Die Benutzerverbindungsnachricht muß in diesem Bereich linksbündig stehen.

-> int *x_datal
Zeiger auf eine Längenangabe (wie bei x_datarq), aber mit der Einschränkung, daß diese Angabe niemals größer als X_EXP_SIZE sein kann.

Fall x_opt = NULL (siehe x_opt):
*x_datal entspricht genau der Länge der zu sendenden Daten aus x_datap.
Maximale Angabe in *x_datal = X_EXP_SIZE.
Maximum von Benutzerdaten pro x_xdatrq: X_EXP_SIZE.
Minimum von *x_datal (1Byte Sendedaten): *x_datal = 1.

Fall x_opt != NULL (siehe x_opt):
*x_datal muß um X_DRQPHL größer angegeben werden, als es den zu sendenden Daten entspricht.
Maximale Angabe in *x_datal = X_EXP_SIZE.
Maximum von Benutzerdaten pro x_xdatrq:
X_EXP_SIZE - X_DRQPHL.
Minimum von *x_datal (1Byte Sendedaten):
*x_datal = 1 + X_DRQPHL.

-> union {struct x_optd1 optd1;} *x_opt
Zeiger auf eine Variante, die die Struktur x_optd1 oder die Angabe NULL enthält. Die Angabe NULL ist obligatorisch, wenn beim Verbindungsaufbau vereinbart wurde, daß in der Datenphase ohne NEABX-Protokoll gearbeitet wird.
Die Struktur x_optd1 ist in der Datei neabx.h definiert.

```
  struct x_optd1 {
->        int x_optnr;          /* Optionsnummer */
->        int x_code;           /* Nachrichtencode */
->        int x_strukt;         /* Nachrichtenstruktur */
  };
```

x_optnr
Optionsnummer. Anzugeben ist X_OPTD1.

x_code
bezeichnet den Nachrichtencode. Anzugeben ist X_TRANS, die zu sendenden Vorrangdaten sind transparent.

x_strukt
Nachrichtenstruktur. Anzugeben ist X_ETX. Vorrangdaten können nur ungeblockt gesendet werden.

Ergebnis

X_OK
Aufruf erfolgreich, Sie können weiter senden.

X_XDATSTOP
Aufruf erfolgreich, Sie dürfen aber erst weitere Vorrangdaten senden, wenn das Ereignis X_XDATGO oder X_DATAGO eingetroffen ist.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Beispiel

Siehe Beispiel für x_datarq().

Vorrangdatenfluß stoppen - expedited data stop

x_xdatstop() sperrt den Vorrangdatenfluß auf der angegebenen Verbindung. Sie teilen NEABX mit, daß Sie nicht bereit sind, für diese Verbindung Vorrangdaten zu empfangen. Ein bereits angezeigtes Ereignis X_XDATIN müssen Sie zuvor noch entgegennehmen.

Dieser Aufruf ist nur bei Rechnerkopplung sinnvoll.

Der Aufruf bewirkt im einzelnen:

- Die lokale TS-Anwendung erhält das Ereignis X_XDATIN und X_DATAIN nicht mehr zugestellt. Sie können aber währenddessen andere NEABX-Funktionen aufrufen, z.B. eine weitere Verbindung aufbauen, über die Sie die ankommenden Vorrangdaten weitergeben wollen.
- Die sendende TS-Anwendung erhält beim Aufruf x_xdatrq() für diese Verbindung das Ergebnis X_XDATSTOP, beim Aufruf x_datarq() das Ergebnis X_DATASTOP. Sie darf dann weder Vorrangdaten noch normale Dateneinheiten senden.

Erneut freigeben können Sie den Vorrangdatenfluß mit x_xdatgo().

```
int x_xdatstop(tref)
```

Parameter

-> int *tref
Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, für die Sie den Vorrangdatenfluß stoppen wollen.

Ergebnis

X_OK
Der Aufruf war erfolgreich.

X_ERROR
Fehler. Fehlercode mit x_error() abfragen.

Hinweis

Wenn Sie bei x_event() das Ereignis X_XDATIN erhalten haben, müssen Sie die anstehende Vorrangdateneinheit vollständig lesen. x_xdatstop() bewirkt nur, daß kein weiteres Ereignis X_XDATIN oder X_DATAIN zugestellt wird.

Beipiel

```
#include        <cmx.h>
#include        <neabx.h>
#include        <tsnx.h>
#include        <stdio.h>
    .
    .
int   tref;       /* Transportreferenz */
    .
    .
if (x_xdatstop(&tref) == X_ERROR)
    fprintf(stderr, "Fehler 0x%x bei x_xdatstop für tref %x\n",
        x_error(), tref);
```

Der Aufruf stoppt den Vorrangdatenfluß für die mit tref angegebene Verbindung.

7 Allgemeines NEABV-Protokoll

7.1 NEABV-Protokoll für die Kommunikation über ICMX (NEA)

Soll Ihre CMX-Anwendung mit einer bestehenden Anwendung im TRANSDATA-Netz kommunizieren, so müssen Sie beim Verbindungsaufbau über die Migrationsschnittstelle ICMX(NEA) das Benutzerdienst-Verbindungsprotokoll (NEABV, SIEMENS Norm SN 77303) einhalten.

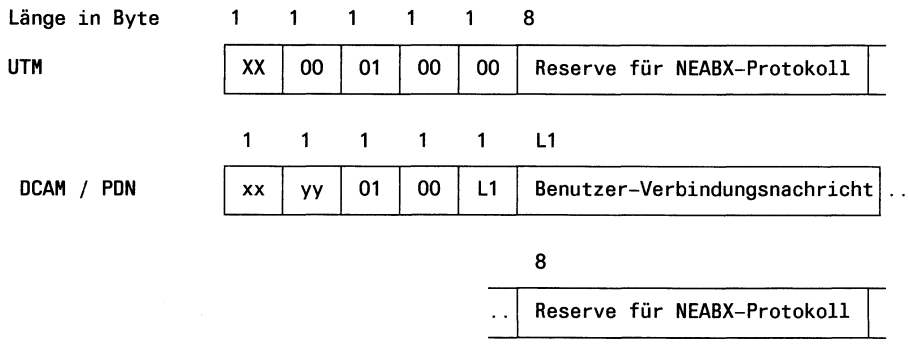
Bestehende Anwendungen im TRANSDATA-Netz können sein

- UTM-Anwendungen (aus UTM-Sicht: PTYPE = APPLI)
- DCAM-Anwendungen (aus DCAM-Sicht: EDIT = USER)
- PDN-Anwendungen (aus PDN-Sicht: Partnercharakteristik bei YOPNCON = Anwendung)

Die folgenden praktischen Hinweise sollen eine protokollgerechte Programmierung des Verbindungsaufbaus über ICMX(NEA), ohne genauere Kenntnis der Norm, ermöglichen.

Das NEABV-Protokoll wird beim Verbindungsaufbau in Form strukturierter Benutzerdaten übertragen.
Bei den Aufrufen x_conrq() und x_conrs() muß die CMX-Anwendung das NEABV-Protokoll vor der Benutzer-Verbindungsnachricht in den Datenpuffer eintragen. Bei den Aufrufen x_conin() und x_concf() steht das NEABV-Protokoll vor der Benutzer-Verbindungsnachricht im Datenpuffer.

Aufbau der Benutzerdaten im Datenpuffer x_udatap bei Rechnerkopplung:



xx: Vereinbarung über das Benutzerdienstprotokoll in der Datenphase

x_conrq, x_conrs, x_concf: xx = X'01'
d.h. kein Benutzerdienstprotokoll in der Datenphase.

x_conin: xx = X'00' oder X'01'
d.h. die Partneranwendung richtet sich nach der Festlegung durch die CMX-Anwendung (xx = X'00') oder schlägt von sich aus den Verzicht auf ein Benutzerdienstprotokoll in der Datenphase vor (xx = X'01').

yy: Vereinbarung über die Initiative bei der Datenübermittlung

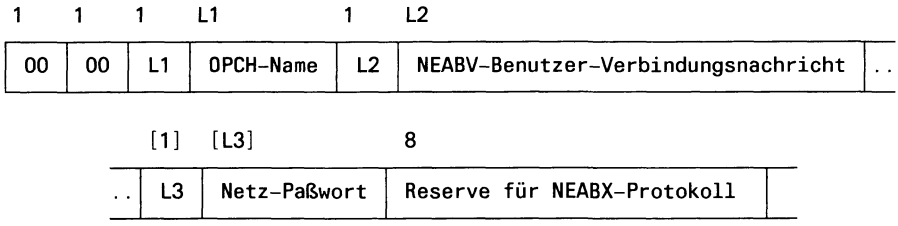
yy = X'01': Der Absender wird mit der Datenübermittlung beginnen.

yy = X'00': Keine Angabe bzw. Einverständnis mit dem Vorschlag des Kommunikationspartners.

L1: Länge der folgenden Benutzer-Verbindungsnachricht
X'00' <= L1 <= X'50' d.h. die folgende Benutzer-Verbindungsnachricht kann zwischen 0 und 80 Zeichen lang sein.

Die Reserve für das NEABX-Protokoll (8Byte) ist für die CMX-Anwendung ohne Bedeutung, sie muß jedoch bei der Reservierung des Pufferbereiches berücksichtigt werden und in der Längenangabe x_udatal enthalten sein.

Aufbau der Benutzerdaten im Datenpuffer x_udatap bei Stationskopplung:



L1: Länge des OPCH-Namens (X'00' < = L1 < = X'08')

OPCH-Name: Name der benutzten Operation-Characters. Dieser Name ist in der PDN-Generierung festgelegt. Geben Sie keinen Namen an, so wird der im PDN festgelegte Standardwert verwendet.

L2: Länge der Benutzer-Verbindungsnachricht
X'00' < = L2 < = X'50'
d.h. die folgende Benutzer-Verbindungsnachricht kann zwischen 0 und 80 Zeichen lang sein.

L3: Länge des Netz-Paßwortes (X'00 < = L3 < = X'08')

Netz-Paßwort: Das Netzpaßwort wird ab PDN V10.0 bei Stationsanschluß benötigt, um ggf. eine Berechtigungsprüfung der Datenstation (bzw. der über Stationskopplung angeschlossenen CMX-Anwendung) für den Zugang zu anderen Rechnern im TRANSDATA-Netz zu ermöglichen.

Hinweis

Für die Länge des Netzpaßwortes (L3) und für das Netzpaßwort selbst gilt: bei x_conrq und x_conrs von der CMX-Anwendung einzutragen. Bei x_conin und x_concf nicht vorhanden.

Die Reserve für das NEABX-Protokoll (8 Byte) ist für die CMX-Anwendung ohne Bedeutung, sie muß jedoch bei der Reservierung des Pufferbereiches berücksichtigt werden und in der Längenangabe x_udatal enthalten sein.

7.2 Der NEABV-Service

Dem Benutzer wird ein Dienst zur Verfügung gestellt, durch den das NEABV-Protokoll erzeugt wird, um Fehlbedienung bei der Erstellung des Protokolls und die damit bewirkten Rechner-Abstürze zu vermeiden.

Das NEABV-Protokoll ist bei folgenden NEABX-Aufrufen einzufügen:

- Connection Request (x_conrq)
- Connection Response (x_conrs)

und wird angeliefert bei:

- Connection Indication (x_conin)
- Connection Confirmation (x_concf)

Die zwei nachfolgenden Aufrufe dienen zur Erzeugung und zur Analyse des NEABV-Protokolls. Sie können jeweils vor bzw.nach den o.a. x_conxx-Aufrufen zur Erzeugung bzw. Analyse des NEABV-Protokolls abgesetzt werden.

- x_neavo(), Erzeugen des NEABV-Protokolls bei Ausgabe.
- x_neavi(), Analysieren eines ankommenden NEABV-Protokolls.

7.2.1 Erzeugen des NEABV-Protokolls (bei PC-Ausgabe)

x_neavo Erzeugen des NEABV-Protokolls in einem Speicher, der anschließend dem Aufruf x_conrq bzw. x_conrs zur Verfügung gestellt werden kann.
Die dazu nötigen Parameter werden in der Struktur x_opt versorgt.

```
int x_neavo(x_udatap,x_udatal,x_opt)
```

Parameter

<- char *x_udatap
Zeiger auf einen Bereich, der von der Funktion mit den NEABV-Protokolldaten versorgt wird.
Die Größe des Bereiches wird beim Aufruf der Funktion in x_udatal angegeben.
Bei Rückkehr enthält die Längenangabe zu x_udatap den aktuellen Wert der NEABV-Nachricht plus der für das NEABX-Protokoll benötigten Reserve.

<> int *x_udatal
Zeiger auf einen Bereich, der die Länge des x_udatap-Bereiches beschreibt:
Bei Funktionsaufruf: Länge des zur Verfügung gestellten x_udatap-Bereiches und
bei Funktionsrückkehr: Benutzte Länge plus nötige Reserve für den NEABX-Header, der vom Aufrufer erst bei x_conrq bzw. x_concf hinzugefügt werden muß.
Die maximal benötigte Länge beträgt unter Berücksichtigung der Reserve für den NEABX-Header (8 Byte) 109 Byte (i.e. bei Stationskopplung und Nutzung des Netzpaßwortes). Siehe auch unter x_opchl, x_bvmsgl und x_npwl.

```
-> union {struct x_optrk optrk;
         struct x_optsk optsk;} *x_opt
```

Zeiger auf die Struktur x_optrk bei Rechnerkopplung oder x_optsk bei Stationskopplung.
Die Optionsnummer der Struktur läßt erkennen, um welche Struktur es sich handelt.
Die Strukturen sind in der Datei neabx.h definiert.

```
struct    x_optrk   {   /* STRUKTUR x_opt BEI RECHNERKOPPLUNG */
->  int     x_optnr;     /* Optionsnr. = X_OPTRK               */
->  int     x_init;      /* Initiative bei Datenübermittlung   */
    int     x_opchl;     /* bei Ausgabe nicht relevant         */
    char   *x_opchp;     /* bei Ausgabe nicht relevant         */
->  int     x_bvmsgl;    /* Länge d. Benutzer-Verb.-Nachricht  */
->  char   *x_bvmsgp;};  /* Zeiger auf Benutzer-Verb.-Nachricht*/

struct    x_optsk   {   /*STRUKTUR x_opt BEI STATIONSKOPPLUNG */
->  int     x_optnr;     /* Optionsnr. = X_OPTSK               */
->  int     x_opchl;     /* Länge d. OPCH in x_opchp           */
->  char   *x_opchp;     /* Pointer auf OPCH-Name              */
->  int     x_bvmsgl;    /* Länge d. Benutzer-Verb.-Nachricht  */
->  char   *x_bvmsgp;    /* Zeiger auf Benutzer-Verb.-Nachricht*/
->  int     x_npwl;      /* Länger des Netzpasswortes          */
->  char   *x_npwp;   };  /* Zeiger Netzpw. (nur bei x_conrq !) */
```

-> int x_optnr
Optionsnummer: Anzugeben ist X_OPTRK für Rechnerkopplung oder X_OPTSK für Stationskopplung

-> int x_init
Initiative zur Datenübertragung bei Rechnerkopplung.
Anzugeben ist X=MYINIT oder X_INITRQ.
Wenn andere als die vorgesehenen Werte angegeben werden, wird ein Fehler gemeldet (siehe auch Kapitel Erläuterungen zur Verwendung der Parameter).

-> char *x_opchp
Stationskopplung: Zeiger auf einen Bereich, der den OPCH-Namen enthält.
Die gefüllte Länge wird durch x_opchl beschrieben.
Rechnerkopplung: ohne Bedeutung

-> int x_opchl
Stationskopplung: Längenangabe zu dem Bereich x_opchp.
Maximale Länge ist 8.
Rechnerkopplung: ohne Bedeutung

-> char *x_bvmsgp
Zeiger auf einen Bereich, der die NEABV-Benutzer-Verbindungsnachricht bei Rechner- und Stationskopplung enthält.
Die gefüllte Länge wird durch x_bvmsgl beschrieben.

-> int x_bvmsgl
Längenangabe zu dem Bereich x_bvmsgp.
Maximale Länge ist 80.

-> char *x_npwp
Zeiger auf einen Bereich, der das Netzpaßwort bei Stationskopplung enthält.
Die gefüllte Länge wird durch x_npwl beschrieben.
Das Netzpaßwort wird nur bei x_conrq benötigt.
Das Netzpaßwort wird erst ab PDN V10.0 unterstützt.

-> int x_npwl
Längenangabe zu dem Bereich x_npwp.
Maximale Länge ist 8.
Das Netzpaßwort wird erst ab PDN V10.0 unterstützt.

7.2.2 Analyse des NEABV-Protokolls (bei PC-Eingabe)

x_neavi Analyse des NEABV-Protokolls aus einem Datenbereich, der durch die Aufrufe x_conin bzw. x_concf vom Netz aus versorgt wurde.
Die Analyse-Ergebnisse werden in Elementen der durch x_opt gezeigten Struktur abgelegt.

Der durch x_conin bzw. x_conrs gelieferte Zeiger kann an x_neavi übergeben werden, wo die Daten in die Struktur x_opt übertragen werden.

```
int x_neavi(x_udatap,x_udatal,x_opt)
```

Paramter

-> char *x_udatap
Zeiger auf einen Bereich, in dem die von x_conin bzw. x_concf empfangenen NEABV-Protokolldaten stehen, die von x_neavi analysiert und in die in x_opt angegebene Struktur gebracht werden sollen.
Es ist sinnvoll den von x_concf/x_conin gelieferten x_updatap weiterzureichen.

-> int *x_udatal
Zeiger auf einen Bereich, der die Länge des x_udatap-Bereiches beschreibt.
Es ist sinnvoll den von x_concf/x_conin gelieferten x_udatal weiterzureichen.

<> union {struct x_optrk optrk;
struct x_optsk optsk;} *x_opt

Zeiger auf einen Bereich, der nach Analyse der in x_udatap übergebenen NEABV-Nachricht die zu Rechner- bzw. Stationskopplung gehörenden Ergebnisse enthält.
Bei Rechnerkopplung beschreibt die Struktur x_optrk und bei Stationskopplung die Struktur x_optsk die Ergebnisse.
Der erste Wert der Struktur läßt erkennen, um welche Struktur es sich handelt.

```
struct    x_optrk   {   /* STRUKTUR x_opt BEI RECHNERKOPPLUNG  */
->  int    x_optnr;     /* Optionsnr. = X_OPTRK                */
<-  int    x_init;      /* Initiative bei Datenübermittlung    */
<-  int    x_opchl;     /* Länge d. OPCH in x_opchp            */
<-  char  *x_opchp;     /* Pointer auf OPCH                    */
<-  int    x_bvmsgl;    /* Länge d. Benutzer-Verb.-Nachricht   */
<-  char  *x_bvmsgp;};  /* Zeiger auf Benutzer-Verb.-Nachricht */

struct    x_optsk   {   /*STRUKTUR x_opt BEI STATIONSKOPPLUNG  */
->  int    x_optnr;     /* Optionsnr. = X_OPTSK                */
<-  int    x_opchl;     /* Länge OPCH: irrelev. b. Stat.kopplg.*/
<-  char  *x_opchp;     /* Point  → OPCH: irrelev.b.St.kopplg. */
<-  int    x_bvmsgl;    /* Länge d. Benutzer-Verb.-Nachricht   */
<-  char  *x_bvmsgp;    /* Zeiger auf Benutzer-Verb.-Nachricht */
    int    x_npwl;      /* ankommend irrelev. (x_concf,x_conin)*/
    char  *x_npwp;  };  /* ankommend irrelev. (x_concf,x_conin)*/
```

-> int x_optnr
Optionsnummer: Anzugeben ist X_OPTRK für Rechnerkopplung oder X_OPTSK für Stationskopplung.

<- int x_init
Initiative zur Datenübertragung bei Rechnerkopplung.
Der Wertebereich ist X_MYINIT oder X_INITRQ.

<- char *x_opchp
Rechnerkopplung: Zeiger auf einen Bereich, der die Operations-Characters enthält.
Die aktuelle Länge wird durch x_opchl beschrieben.
Stationskopplung: ohne Bedeutung.

<- int x_opchl
Rechnerkopplung: Längenangabe zu dem Bereich x_opchp.
Stationskopplung: ohne Bedeutung.

<- char *x_bvmsgp
Zeiger auf einen Bereich, der die NEABV-Benutzer-Verbindungsnachricht bei Rechner- und Stationskopplung enthält.
Die aktuelle Länge wird durch x_bvmsgl beschrieben.
Die Nachricht wird von x_neavi nicht umcodiert oder behandelt.

<- int x_bvmsgl
Längenangabe zu dem Bereich x_bvmsgp.
Maximale Länge ist 80.

char *x_npwp
Irrelevant bei Empfang des NEABV-Protokolls.
Bei x_concf und x_conin wird kein Netzpaßwort geliefert. Der Wert ist mit NULL belegt.

int x_npwl
Irrelevant bei Empfang des NEABV-Protokolls.
Bei x_concf und x_conin wird kein Netzpaßwort geliefert. Der Wert ist mit NIL belegt.

7.2.3 Erläuterungen zur Verwendung von x_init

x_init Initiative bei der Datenübermittlung

Möglichke Werte:

X_MYINIT (X'01'): Vorschlag mit der Datenübermittlung zu beginnen.

X_INITRQ (X'00'): Warten auf Partnervorschlag (gelegentlich als NOINIT bezeichnet).*)

X_INITOK (X'00'): Einverständnis mit dem Partnervorschlag.*)

Anmerkung
*) Die Werte sind gleichcodiert.

- Rechnerkopplung:
  Beschreibung der x_init-Angaben des rufenden Partners in x_conrq bzw. wie sie vom gerufenen Partner in x_conin empfangen werden können.
  Die erwartete Anwort x_init wird vom gerufenen Partner in x_conrs gegeben bzw. in x_concf vom rufenden Partner empfangen.

  – x_init = X_MYINIT
    Vorschlag des rufenden Partners, daß er mit der Datenübermittlung beginnen will.
    Antwort des gerufenen Partners: x_init = X_INITOK (Einverständnis mit Partnervorschlag) oder x_disrq im Falle keines Einverständnisses.

  – x_init = X_INITRQ
    Partnervorschlag erwartet.
    Erwartete Anwort des gerufenen Partners: x_init = X_MYINIT (INIT des gerufenen Partners) oder x_disrq, andernfalls soll der rufende Partner die Verbindung mit x_disrq abbrechen, weil es zu keiner Vereinbarung gekommen ist.

    Anmerkung
    Bei x_init = X_INITOK als Antwort besteht die Gefahr von Gegensprechen bzw. ewigem Warten. Deshalb sollte dann mit x_disrq geantwortet werden.

- Stationskopplung:
  Die in x_conrq und x_conrs gemachten Angaben sind irrelevant, weil das PDN die x_init Angaben bereits beim Verbindungsaufbau mit dem Partner ausgehandelt hat.

  Um einen definierten Wert festzulegen, ist x_init = X_INITRQ zu setzen. Dies erfolgt bei der Verwendung von x_neavo automatisch.

8 Was sonst noch wichtig ist

8.1 CMX installieren

CMX installieren Sie wie andere Softwareprodukte in SINIX, am zweckmäßigsten unter der Benutzerkennung **admin** unter der Auswahl des Menüs Systemverwaltung.
Vor der Installation von CMX sollte eines der Produkte CCP [CCP] installiert worden sein. Beachten Sie zur Installation von CMX und CCP auch die Freigabemitteilung der entsprechenden Produkte.
Nachdem Sie CCP und CMX installiert haben, müssen Sie für die Kommunikationspartner - sowohl die lokalen als auch die entfernten - entsprechende Einträge im TS-Directory vornehmen. Dazu dient das Erfassungsprogramm TNSADMIN [CCP]. Erst wenn Sie diese Erfassung erfolgreich beendet haben, können Sie selbst entwickelte CMX-Anwendungen oder Systemanwendungen, die auf CMX aufsetzen, zum Ablauf bringen.

8.2 Übersetzen und Binden, Kompatibilität

Übersetzen und Binden

Wenn Sie selbst CMX-Anwendungen entwickeln wollen, können Sie mit Hilfe dieses Manuals entsprechende C-Programme editieren. Dann brauchen Sie das C-Entwicklungssystem [CES] von SINIX, um diese C-Programme zu übersetzen (compilieren) und darin die CMX-Funktionen aus der CMX-Bibliothek einzubinden.
Für die einzelnen PCs ist der Aufruf des C-Compilers mit eingeschlossener Bindephase im jeweiligen Universum wie folgt:

```
cc -o prog prog.c ... -lcmx
```

Kompatibilität

Die mit diesem Manual erstellten CMX-Anwendungsprogramme sind objektkompatibel unter den SINIX-Rechnern austauschbar, sofern auf diesen SINIX ab V2.1 und CMX ab V2.0 abläuft. Dabei muß die Konfiguration der Kommunikations-Hard- und -Software (CCs und CCPs) vergleichbar sein.

Beachten Sie die Freigabemitteilungen der entsprechenden Produktversionen.

8.3 Einschränkungen bei Stationskopplung

Dieser Abschnitt enthält Einschränkungen bei der Verwendung der Stationsanbindung der SINIX-Rechner. Diese Einschränkungen müssen Sie beachten, wenn Sie portable CMX-Anwendungen (bzgl. Stations- und Rechneranbindung) schreiben wollen. Beachten Sie auch, daß die Transportsysteme für die Stationsanbindung [CCP] gewisse System- und Benutzeroptionen nicht bieten (siehe hierzu Abschnitt 1.2.4 und 8.4).

An der Schnittstelle ICMX(L) existieren mit CCP-STA1 ab V2.1 folgende funktionelle Einschränkungen, die auch von CMX-Anwendungen über eine Rechneranbindung beachtet werden sollten, wenn man portable CMX-Anwendungen (bzgl. Stations- und Rechneranbindung) anstrebt. Die Einschränkungen beruhen darauf, daß das eigentliche Transportsystem entweder ganz entfällt (primary) oder im Anschlußrechner liegt.

- t_conrq()

  Jede CMX-Anwendung kann grundsätzlich zu einem Zeitpunkt nur eine Verbindung über Stationskopplung halten, unabhängig davon, wie sie in Prozesse organisiert ist. Wie bei der Rechnerkopplung kann sich nun aber (im Gegensatz zur V2.0) ein Prozeß gleichzeitig zu mehreren Anwendungen anmelden.
  Reaktion bei Fehlbedienung: T_DISIN mit Reason 448

- t_conrq()

  Über CCP-STA1 ist grundsätzlich kein Transfer von Vorrangdaten möglich. Eventuelle Auswahl von Vorrangdaten beim Aufruf t_conrq() wird beim t_concf() automatisch "heruntergehandelt".

- t_conrs()

  Bei der Verbindungsbeantwortung (t_conrs()) dürfen keine Benutzerdaten übergeben werden.
  Reaktion bei Fehlverhalten: Verlust der Benutzerdaten

- t_datarq()

  Bei der Wahl von CCP-STA1/SDLC ist die Gesamtlänge einer Nachricht auf eine Dateneinheit (TIDU) begrenzt.
  Ansonsten ist die Gesamtlänge einer Nachricht auf 3064 Byte im normierten Übertragungsmodus und auf 4088 Byte im transparenten Modus festgelegt. Zur Begriffsbildung sei auf das CCP-STA1-Manual verwiesen.

Die maximale Länge einer Dateneinheit (TIDU) ist dynamisch durch t_info() erfragbar (z.Zt. 440 Byte). Es dürfen maximal 10 TIDUs mit dem T_MORE-Flag miteinander verkettet werden.
Reaktion bei Fehlverhalten: T_DISIN mit Reason 465

- t_datarq()

  Bei einer Einbindung in das TRANSDATA-Netz sollte nach dem Verbindungsaufbau, bei dem vorzugsweise die lokale CMX-Anwendung der rufende Partner sein sollte, auch der rufende Partner die Initiative beim Datenaustausch übernehmen. Andernfalls kann es bei hoher Last im Anschlußrechner zum Mißerfolg beim Verbindungsaufbau kommen.

- t_datain(), t_datastop()

  Eine empfangsseitige Flußregelung ist rein lokal. Sie beeinflußt nicht das Sendeverhalten des Partners. Deshalb werden auf einer Verbindung empfangene und nach einer gewissen Zeit nicht abgeholte Daten vernichtet, um den übrigen Transfer auf der Leitung zu ermöglichen. Die Zeit, nach der Empfangsdaten verworfen werden, kann bei CCP-STA1/MSV1 über den WABT-Zähler bei der Konfigurierung des CCP eingestellt werden (genaueres siehe CCP-STA1-Manual).

- t_disrq(), t_disin()

  Beim Verbindungsabbau dürfen keine Benutzerdaten übergeben werden.
  Reaktion bei Fehlverhalten: Verlust der Benutzerdaten

- Namensraum

  Aus der Sicht des Netzes ist der Namensraum der über CCP-STA1 erreichbaren CMX-Anwendungen durch die Generierung als Stationen im Anschlußrechner beschränkt.

Bei der Verwendung von CCP-STA2 (BAM) oder CCP-STA1 bis zu V2.0 einschließlich, gelten **zusätzlich** zu den oben beschriebenen noch weitergehende Einschränkungen:

- t_conrq()

  Jede CMX-Anwendung kann grundsätzlich nur eine Verbindung halten, unabhängig davon, wie sie in Prozesse organisiert ist. Jeder Prozeß kann sich **höchstens für eine** CMX-Anwendung anmelden. Insbesondere kann damit **nicht** in einem Prozeß gleichzeitig eine Stations- und Rechnerkopplung verwaltet werden. Beim Anmelden wird durch den LOKALEN NAMEN definitiv festgelegt, ob der Prozeß für Stations- oder Rechnerkopplung tätig wird.
  Obwohl sich mehrere Prozesse für dieselbe CMX-Anwendung anmelden können, kann immer nur einer die eine Verbindung halten. Alle anderen Prozesse können nur Verbindungsumlenkungen entgegennehmen.

- t_conrq()

  Die Überwachungszeit für die Verbindung (Parameter t_timeout) muß 0 sein.

- t_datarq()

  Die Gesamtlänge einer Nachricht ist beschränkt. Sie kann die Länge **einer** Dateneinheit (TIDU) nicht überschreiten, d.h. der chain-Parameter beim t_datarq muß immer mit T_END versorgt werden.
  Die maximale Länge einer Dateneinheit ist dynamisch mit t_info() erfragbar. Sie beträgt z.Zt. bei der Verwendung von CCP-STA2 3060 Byte, bei CCP-STA1/MSV1 V2.0 3072 Byte und bei CCP-STA1/HDLC V2.0 3060 Byte.

  Beachten Sie bitte die Unterschiede zu den entsprechenden Aussagen zu CCP-STA1 V2.1 (s.o.)!

- t_datain(), t_datastop()

  Wie weiter oben schon beschrieben, ist die empfangsseitige Flußregelung rein lokal. Bei CCP-STA1/HDLC V2.0 und CCP-STA2 werden nicht abgeholte Empfangsdaten nach etwa 25 sec vernichtet. Bei CCP-STA1/MSV1 V2.0 kann diese Zeit über den WABT-Zähler eingestellt werden. Im Falle CCP-STA2 wird darüberhinaus ein Dämonprozeß (bamdae) aktiviert, der den Prozeß, welcher die Verbindung bis dahin gehalten hat, beendet und die Verbindung abbaut. Näheres finden Sie in den entsprechenden CCP-Manualen.

8.4 Systemoptionen

Die Verfügbarkeit der Systemoptionen (siehe Abschnitt 1.2.4) Benutzerdaten und Vorrangdaten für die über CMX verwendbaren Transportsysteme ergibt sich aus folgender Tabelle. Die linke Spalte enthält das betroffene Transportsystem mit seinem Produktnamen [CCP]. Ziffern, die unmittelbar von einer runden Klammer gefolgt werden (z.b. "4)10)"), beziehen sich auf die unten stehenden Anmerkungen.

CCP-	Benutzerdaten 1) in Byte bei t_conrq() t_conin() t_concf()	Benutzerdaten 1) in Byte bei t_conrs()	Benutzerdaten 1) in Byte bei t_disrq() t_disin()	Vorrang-daten-länge in Byte	Länge der Nachricht	Anzahl Daten-einheiten pro Nachricht
LAN2	32 5)	32 7)	64 8)	16 3)10)	beliebig	beliebig
STA[1-2]	80	0	0	0	3064 12) 4088 12)	10 12)
WAN[1-2]	92 5)	92 7)	1 9)	12 3)4)10)	belieb.13)	beliebig 13)
WAN3	32 6)	0 8)	1 8)	1 3)11)	beliebig	beliebig
WAN[4-5]	0 8)	0 8)	0 8)	0 2)	beliebig	beliebig
WAN6	32 5)	32 7)	64 8)	16 3)10)	beliebig	beliebig

Anmerkung

1) Die Benutzerdaten beim Verbindungsaufbau enthalten Nettobenutzerdaten und Migrations-Protokolle; die Tabelle gibt nur die Nettobenutzerdaten an, also nur das, was die Partner austauschen können; mit Migrationsprotokollen dürfen die Benutzerdaten T_MSG_SIZE Byte nicht überschreiten.

2) Beim Verbindungsaufbau veranlaßt das Transportsystem die Vereinbarung, daß Vorrangdatenaustausch während der Kommunikation nicht zugelassen ist; deshalb weist CMX den t_xdatrq() mit (T_ERROR und) T_WSEQUENCE zurück.

3) Falls die Partner beim Verbindungsaufbau vereinbart haben, daß Vorrangdatenaustausch bei der Kommunikation nicht zugelassen ist, weist CMX den t_xdatrq() mit (T_ERROR und) T_WSEQUENCE zurück.

4) Die Zulassung von Vorrangdaten wird beim Verbindungsaufbau nur lokal vereinbart, und zwar so, wie es der lokale Aufrufer wünscht. Es kann also vorkommen, daß der rufende Partner Vorrangdaten zuläßt, der gerufene aber nicht. Wenn in diesem Fall der rufende Partner mit t_xdatrq() Vorrangdaten absendet, so erfolgt ein Verbindungsabbau durch CMX. Der sendende Partner erhält den reason 389, der empfangende Partner den reason 464 (siehe Abschnitt 6, t_disin()).

5) Wenn die Benutzerdaten zu lang sind, wird entweder der t_conrq() von CMX mit (T_ERROR und) T_WPARAMETER abgewiesen oder das lokale Transportsystem bricht die Verbindungsaufbauphase mit reason 470 ab.

6) Wie 5), jedoch mit reason 483.

7) Wenn die Benutzerdaten zu lang sind, wird entweder der t_conrs() von CMX mit (T_ERROR und) T_WPARAMETER abgewiesen oder das lokale Transportsystem bricht die Verbindungsaufbauphase mit reason 470 für den rufenden Partner und reason 386 für den gerufenen Partner ab.

8) Wenn die Benutzerdaten zu lang sind, wird entweder der t_conrq() bzw. t_conrs() bzw. t_disrq() von CMX mit (T_ERROR und) T_WPARAMETER abgewiesen oder das lokale Transportsystem schneidet den überzähligen Anteil der Benutzerdaten ab und führt den Auftrag mit den verbleibenden Daten aus.

9) Wenn die Benutzerdaten zu lang sind, wird entweder der t_disrq() von CMX mit (T_ERROR und) T_WPARAMETER abgewiesen oder das Transportsystem führt den Auftrag ohne Benutzerdaten aus.

10) Wenn die Vorrangdaten zu lang sind, wird entweder der t_xdatrq() von CMX mit (T_ERROR und) T_WPARAMETER abgewiesen oder das lokale Transportsystem baut die Verbindung ab. Der sendende Partner erhält reason 470, der empfangende reason 256.

11) Wenn die Vorrangdaten zu lang sind, wird entweder der t_xdatrq() von CMX mit (T_ERROR und) T_WPARAMETER abgewiesen oder das lokale Transportsystem schneidet den überzähligen Anteil der Vorrangdaten ab und führt den Auftrag mit den verbleibenden Daten aus.

12) Die Angaben gelten für CCP-STA1 ab V2.1 für den normierten Modus (3064) und transparenten Modus (4088).
Bei CCP-STA1 V2.0 gilt einheitlich 3060, desgleichen für CCP-STA2.

Die Anzahl der Dateneinheiten pro Nachricht gilt nur für CCP-STA1 ab V2.1. Für CCP-STA1 V2.0 und CCP-STA2 beträgt sie 1.

13) Die Angabe gilt nur für homogene Kopplungen von SINIX-Rechnern. Bei Kopplungen zu PDN oder BS2000 ist die Nachricht auf 4096 Byte oder maximal 16 Dateneinheiten begrenzt.

8.5 Die Include-Dateien cmx.h, tnsx.h und neabx.h

Im Anhang sind die Include-Dateien cmx.h, tnsx.h und neabx.h aufgelistet, so wie sie zum Zeitpunkt der Erstellung dieses Manuals vorlagen. Sie können Sie für gelegentliche Referenzen bei der Programmierung verwenden.

Auf Ihrem PC finden Sie diese Dateien im Dateisystem unter

```
/usr/include/cmx.h
/usr/include/tnsx.h
/usr/include/neabx.h
```

Im Zweifelsfall (bei eventuellen Abweichungen) gilt stets der Inhalt dieser Datei. Sie wird bei der Compilierung Ihres CMX-Anwendungsprogrammes verwendet.

8.6 Verfolgerinformationen für CMX

CMX sammelt auf Anforderung die unten aufgeführten Verfolgerinformationen. Sie dienen dazu, bei unerwarteten Fehlermeldungen zu den CMX-Aufrufen über die Auswertung der Verfolgerinformation die Fehlerursache festzustellen.

Die Auswertung erfordert in der Regel Expertenwissen. Im allgemeinen Fall sollten Sie die Verfolgerinformation gemäß der Beschreibung sicherstellen und für die Auswertung durch den Systemkundendienst bereithalten.

Die Verfolgerinformationen liefern Auskunft über

- Rechnerkopplung an WAN und LAN
- Stationskopplung an Anschlußrechner
- Anforderungen an den TNSX

Im Dateiverzeichnis /usr/lib/cmx stehen folgende Programme zur Verfügung, mit denen im Expertenmodus die Verfolger gesteuert bzw. die Verfolgerinformationen sichergestellt werden können:

Programm	Funktion
cmxt	Sicherstellen und Aufbereiten der Traces des CMX-Automaten und der CC-Adapter
cxst	Sicherstellen und Aufbereiten spezifischer Verfolgerinformation bei Stationskopplung
tnsxt	Steuern der Verfolgerinformationen für die Zugriffe zum TNSX
tnsxprop	Ausgabe der Attribute eines Objektes eines TS-Directory

Alle Programme geben ihre Ergebnisse auf die Standardausgabe **stdout**, die (selbsterklärenden) Fehlermeldungen in der Regel auf die Diagnoseausgabe **stderr**, so daß die Sicherstellung der Ergebnisse leicht durch Umlenkung der Ausgabe in eine Datei erreicht werden kann.

Die folgenden Seiten beschreiben die Programme im Detail.

NAME

cmxt - Sicherstellen und Aufbereiten der Traces des CMX-Automaten und der CC-Adapter

SYNOPSIS

/usr/lib/cmx/cmxt [-[bcdesStvx][f datei]]
/usr/lib/cmx/cmxt [-[bcdesStvx]l[1-2][f datei]]
/usr/lib/cmx/cmxt [-[bcdesStvx]w[1-6][f datei]]

BESCHREIBUNG

Die Verfolgereinträge des CMX-Automaten und der CC-Adapter werden im Betriebsystemkern vorgenommen. Sie werden dort in einem dynamisch angelegten Ringpuffer aufgesammelt, solange der jeweilige Verfolger eingeschaltet ist. Das Einschalten des Verfolgers muß vor der Kommunikation vor dem ersten t_attach() der zu überwachenden Prozesse stattfinden. Zu beliebigen Zeitpunkten kann der Verfolger angehalten und der Ringpuffer ausgegeben werden. Mit dem Ausschalten des Verfolgers wird auch der Ringpuffer wieder freigegeben.

cmxt liest den vom Verfolger erzeugten Ringpuffer aus dem Betriebssystem in eine temporäre Datei, wo er in binärem Format zwischengespeichert wird. Von dort erfolgt dann die Druckaufbereitung und die Ausgabe auf stdout. Sie kann in einem komprimierten oder einem ausführlichen Format erfolgen.

Die Argumente haben die im folgenden beschriebene Bedeutung. Bei jedem Aufruf von cmxt kann einer der Schalter -cdesStvx angegeben werden. Ein daran anschließendes l[1-2] oder w[1-6] wählt den Verfolger des CC-LAN-Adapters (l) oder CC-WAN-Adapters (w) für den CC mit der angegebenen Nummer (1-2) oder (1-6) aus. Fehlt diese Angabe, wird der Verfolger des CMX-Automaten ausgewählt. Unmittelbar daran kann noch der Schalter f folgen. Die Schalter bedeuten:

s gewöhnlichen Verfolger (fort)starten;

S erweiterten Verfolger (fort)starten, dient nur für besondere Diagnosefälle;

e Verfolger nur für Fehler (fort)starten, in diesem Fall werden nur fehlerhafte Operationen in den Ringpuffer aufgenommen;

c Verfolger anhalten und Ringpuffer im Binärformat in einer temporären Datei sicherstellen (siehe Schalter **f**);

b Verfolger anhalten und Ringpuffer aufbereiten und abdruckbar im ausführlichen Format nach stdout ausgeben;

d Verfolger anhalten und Ringpuffer aufbereiten und abdruckbar im komprimierten Format nach stdout ausgeben;

t Verfolger abschalten und Ringpuffer freigeben;

v Ringpuffer aus temporärer Datei (siehe Schalter **f**) lesen, im ausführlichen Format aufbereiten und auf stdout ausgeben;

x Ringpuffer aus temporärer Datei (siehe Schalter **f**) lesen, im komprimierten Format aufbereiten und auf stdout ausgeben;

f Der Ringpuffer soll zur späteren Auswertung in **datei**abgespeichert oder zur Aufbereitung aus **datei** gelesen werden. Der Standardwert für **datei** ist cmxt.bin[[lw][1-6]] im laufenden Dateiverzeichnis.

Beispiel

cmxt -s
gewöhnlichen Verfolger des CMX-Automaten (fort)starten;

cmxt -cf cmxt.bin
Verfolger des CMX-Automaten anhalten und Ringpuffer im Binärformat in der Datei cmxt.bin sichern;

cmxt -vf cmxt.bin
Ringpuffer des CMX-Automaten aus cmxt.bin im ausführlichen Format nach stdout aufbereiten;

cmxt -cl1f cxwt.binl1
Verfolger für CC L1 anhalten und Ringpuffer im Binärformat in der Datei cxwt.binl1 sichern;

cmxt -vw2
Verfolger des CC-WAN-Adapters für CC W2 aus Datei cmxt.binw2 (default) lesen und im ausführlichen Format auf stdout aufbereiten.

EXIT STATUS

Bei erfolgreichem Ablauf ist der exit-Status 0, sonst ungleich 0.

FEHLERMELDUNGEN

Folgende selbsterklärende Fehlermeldungen auf stderr sind möglich:

```
"cmxt: Fehler %d bei open von %s\n"
"cmxt: Fehler %d ioctl %x auf %s\n"
"cmxt: Fehler %d bei read von Tracedatei\n"
"cmxt: Fehler %d bei write auf Tracedatei\n"
"cmxt: %s enthält keinen Trace im Binärformat\n"
"cmxt: Unbekanntes Eintragskennzeichen %d im Trace\n"
```

Die protokollierte Fehlernummer (%d) kann über errno.h entschlüsselt werden.

Hinweis

Die mit CMX V2.0 freigegebenen Verfolgerprogramme cmxt, cxlt und cxwt werden in CMX V2.1 durch cmxt realisiert. Die für cxlt und cxwt gültige Syntax und Semantik wird durch cmxt mit unterstützt. cxlt und cxwt sind Shell-Prozeduren, die die alte Syntax mit cmxt realisieren.

NAME

cxst - Sicherstellen und Aufbereiten der Verfolgerinformation bei Stationskopplung

SYNOPSIS

/usr/lib/cmx/cxst [-t] [[-b] pid [dir]]

BESCHREIBUNG

Die Verfolgerfunktionen für die Stationskopplung mit CMX werden beim Ablauf einer CMX-Anwendung durch die Umgebungsvariable **CMXTRACE** gesteuert. Das Einschalten des Verfolgers muß vor der Kommunikation vor dem ersten t_attach() der zu überwachenden Prozesse erfolgen. Es geschieht am besten auf Kommandoebene durch

$ CMXTRACE=-io; export CMXTRACE.

Der Verfolger protokolliert die letzten (mindestens) 100 Ein- und Ausgaben eines Prozesses über Stationskopplung in einem Ringpuffer in den Dateien **usr/tmp/CMXI<pid>** und **/usr/tmpCMXO<pid>** (<pid> ist Prozeß-Id) mit, bis der Prozeß sich beendet und der Verfolger durch

CMXTRACE=""

ausgeschaltet wird.

cxst bereitet den vom Verfolger erzeugten Ringpuffer und ggf. Systemtabellen aus dem Betriebssystemkern abdruckbar auf und gibt sie über stdout aus. Die Schalter und Argumente haben folgende Bedeutung:

t Sicherstellung und Aufbereitung der zum Aufrufzeitpunkt geltenden Tabellen aus dem Betriebssystemkern:

CMX-Kontrolltabelle
CMX-Anwendungstabelle
CMX-Umlenktabelle.

b Bei der Aufbereitung der Verfolgerinformationen aus dem Ringpuffer wird das Bytesplitting rückgängig gemacht. Das Bytesplitting erfolgt für alle Ein- und Ausgaben gewöhnlicher CMX-Anwendungen, nicht aber für Emulationsanwendungen.

pid Prozeßidentifikation des Prozesses, für den der Ringpuffer aufbereitet werden soll. Der Ringpuffer wird in den weiter oben genannten Dateien erwartet.

dir Dateiverzeichnis, aus dem die Dateien CMXI<pid> und CMXO<pid> entnommen werden sollen, Standard: /usr/tmp.

AUSGABEFORMAT

Die Ausgabe des Ringpuffers erfolgt in Zeilen folgenden Formates:

```
IO Nr. hh:mm:ss.mmm lng Text hexadezimal  Text abdruckbar
>    2 13:07:30.840  67 48404030 ...      |H@@0...
                        4f472c30 ...      |OG,0...
```

wobei die einzelnen Einträge folgendes bedeuten:

IO	Aus- (>) oder Eingabe (<) des Prozesses
Nr.	laufende Eintragsnummer modulo 65536
hh:mm:ss.mmm	Zeitstempel mit Stunden (hh), Minuten (mm), Sekunden (ss) und Millisekunden (mmm)
lng	tatsächliche Länge des Textes wie auf der Leitung übertragen (höchstens die ersten 150 Zeichen werden angezeigt)
Text hexadezimal	übertragener Text in Hexadezimaldarstellung
Text abdruckbar	abdruckbare Zeichen des übertragenen Textes

FEHLERMELDUNGEN

Folgende selbsterklärende Fehlermeldungen sind möglich:

```
"Aufruf: cxst [-t] [[-b] pid [dir]]\n"
"cxst: pid %s zu groß, muß kleiner 66000 sein\n"
"cxst: Fehler %d bei open %s\n"
"cxst: Fehler %d bei ioctl %s\n"
"cxst: Fehler %d bei read\n"
"cxst: Interner Programmfehler!\n"
"cxst: Fehler %d in Tracedatei\n"
"cxst: pid in Trace-Datei ungleich pid in Dateinamen\n"
```

Die protokollierte Fehlernummer (%d) kann über errno.h entschlüsselt werden.

NAME

tnsxt - Sicherstellen und Aufbereiten der Verfolgerinformation bei Zugriffen zum Transport Name Service über ICMX(TNS)

SYNOPSIS

/usr/lib/cmx/tnsxt on|off

BESCHREIBUNG

Die Verfolgereinträge für die Zugriffe zum Transport Name Service in SINIX (TNSX) werden an zentraler Stelle pauschal für alle Prozesse vorgenommen. Die letzten 100 Zugriffe zum TNSX werden dort in abdruckbarer Darstellung in einem Ringpuffer aufgesammelt, solange der Verfolger eingeschaltet ist. Der Ringpuffer wird beim Starten des Systems initialisiert und in der Datei **/usr/lib/cmx/tnsxd.trace** (mit link auf DSA_TRACE) angelegt. Er bleibt während der gesamten Lebensdauer des Systems bestehen. Das Einschalten des Verfolgers kann jederzeit erfolgen, muß aber vor den zu überwachenden Zugriffen geschehen. Zu beliebigen Zeitpunkten kann der Verfolger angehalten und der Ringpuffer ausgegeben werden. Bei Fortstart des Verfolgers wird der Ringpuffer fortgesetzt.

tnsxt steuert den Verfolger über das erste Argument. Bei **on**wird der Verfolger (fort)gestartet, bei **off** angehalten.

FORMAT DER VERFOLGEREINTRÄGE

Die Einträge des Verfolgers bestehen ausschließlich aus abdruckbaren Zeichen, so daß der Ringpuffer mit üblichen SINIX-Mitteln aufgelistet werden kann. Im folgenden ist ein Ringpufferauszug angegeben:

```
 1 tnsxd 2.7 87/11/26: (re)starting -- Thu Nov 26 21:39:54 1987
 2         open 1 close 0 read 0 eof 0 write 0 epipe 0
 3 ### CONTROL-FLAG 1: 1   CONTROL-FLAG 2: 0
 4
 5 ### TNS REQUEST 1 (Message Size = 73 Bytes) -- Thu Nov 26 21:52:14 1987
 6 INVOKE-ID: 1570  VERSION: 1  DIRECTORY-ID: 1  REQUEST-TYPE: ReadProperties
 7 PROPERTYLIST: 124  NAMELIST: 0
 8 NAME:
 9    ID: 66004  LNG: 3  VALUE: POL
10 PROPERTIES:
11    ID: 107  TYP: 0
12 RESULT-TYPE: ret_result
13 tnsxd: caught signal 1
14 tnsxd: terminated -- Thu Nov 26 21:52:19 1987
```

Die einzelnen Zeilen haben folgende Bedeutung:

Die ersten beiden Zeilen geben die Version des tnsxd und das Datum des letzten "wrap around" der Verfolgerdatei wieder, gefolgt von einer Statistik aller open-, close-, read- und write-Aufrufe inklusive der read-Aufrufe, die mit EOF bzw. der write-Aufrufe, die mit EPIPE abgeschlossen wurden. Zeile 3 protokolliert den Aufruf, den Verfolger ein- oder auszuschalten. Ab Zeile 5 bis Zeile 12 einschließlich wird eine Anforderung an den TNSX protokolliert. Die Anforderungen sind numeriert und mit einem Zeitstempel versehen. Die Numerierung beginnt mit jedem Start des tnsxd bei 1.

Die einzelnen Felder bedeuten:

Message Size	Länge der Nachricht mit der Anforderung
Zeitstempel	des Beginns der Bearbeitung der Anforderung
INVOKE-ID	Prozeßnummer des aufrufenden Prozesses
VERSION	Versionskennung des Aufrufers
DIRECTORY-ID	betroffenes TS-Directory
REQUEST-TYPE	Typ der Anforderung, korrespondierend mit den Aufrufen von ICMX(TNS)

Die weiteren Zeilen enthalten zusätzliche Parameter der Anforderung, die vom Typ der Anforderung abhängen. Die letzte Zeile eines Eintrags enthält den Ergebnistyp

RESULT-TYPE	mit dem Erfolg der Bearbeitung. ret_result bedeutet positive Bearbeitung, reject (t, c, v) oder ret_error (t, c, v) bedeutet negative Bearbeitung mit Fehlertyp t, -klasse c, -wert v. Sie können mit tnsx.h entschlüsselt werden.

FEHLERMELDUNGEN

Folgende selbsterklärende Fehlermeldungen sind möglich:

```
"\nERROR: unknown parameter\n"
"\nTNS-ERROR: type = %d, class = %d, value = %d\n"
```

Die protokollierten Fehlernummern (%d) können über tnsx.h entschlüsselt werden.

NAME

tnsxprop - Ausgabe der Attribute eines Objektes eines TS-Directory

SYNOPSIS

/usr/lib/cmx/tnsxprop dirid np1 np2 np3 np4 np5

BESCHREIBUNG

tnsxprop gibt in abdruckbarem Format die Werte aller Attribute ("properties") des Objektes mit den angegebenen Namensteilen im TS-Directory dirid auf stdout. Die Argumente np1, np2, np3, np4, np5 entsprechen den Namensteilen 1 bis 5. Für jeden nicht vorhandenen Namensteil muß ein Leerstring ("") übergeben werden.

FORMAT DER AUSGABE VON TNSXPROP

Dem Objekt im TS-Directory 1, das nur Namensteil5 mit Wert "alles" hat, seien alle möglichen Eigenschaften zugeordnet. Dann liefert der Aufruf

```
/usr/lib/cmx/tnsxprop 1 "" "" "" "" alles

EIGENSCHAFTEN DER TS-ANWENDUNG
        1, ../../../../alles

TRANSPORTADRESSE:
     0  02002500 02001000 1B00300F 03490000 |  %       0  I  |
    10  00090001 BA987654 3210FE80 08326E61 |      vT2    2na|
    20  6C326E61 6C                         |12nal           |

MIGRATIONSSERVICE:
        00       00000000

CC-LISTE:
        0200     00000000 00000010

TRANSPORTSYSTEM:
        02       00000010

LOKALER NAME:
     0  01005E00 0E002800 00002000 0800E2C4 |      (         |
    10  C1F24040 40400000 01000800 E6C1D5D5 |  @@@@          |
    20  F1404040 00000800 08005741 4D335741 | @@@      WAM3WA|
    30  4D330000 10000800 57414D34 57414D34 |M3      WAM4WAM4|
    40  00000200 08004C41 4D324C41 4D320000 |      LAM2LAM2  |
    50  40000A00 E3D9C1D5 E2C9E340 8800     |@          @    |
```

Die Ausgabe von Eigenschaften, deren Wert mehr als 4 Byte umfaßt, erfolgt im xd(1)-Format, sonst als Hexaziffernstring, gefolgt von der entsprechenden bitweisen Darstellung, wobei das niederwertigste Bit ganz rechts steht.

EXIT STATUS

Bei erfolgreichem Verlauf ist der exit-Status 0, sonst ungleich 0.

FEHLERMELDUNG

Folgende selbsterklärende Fehlermeldung auf stderr ist möglich:

"tnsxprop: TNS-Fehler (Typ: %d Klasse: %d Wert: %d)"

Die protokollierten Fehlernummern (%d) können über tnsx.h entschlüsselt werden

SIEHE AUCH

tnsxt(CMX)

8.7 Verfolgerinformationen für NEABX

Dynamisch ein- ausschaltbarer NEABX-Trace

Der NEABX-Trace ist über die Umgebungsvariable NEATRACE dynamisch ein- und ausschaltbar. Auch die Umgebungsvariable CMXTRACE schaltet den NEABX-Trace ein, wenn sie gesetzt ist. Die Angaben von NEATRACE sind dominant.

Die Abfrage zum Ein-/Ausschalten des Traces erfolgt nur bei x_attach, in Anlehnung an die Vorgangsweise bei CMX-Trace. Der Trace ist unabhängig von Rechner- oder Stationskopplung ein- bzw. ausschaltbar. Die erzeugte Datei heißt: nea.trc.<pid>.

- Die NEABX-Verfolgerfunktionen für Stations- und Rechnerkopplung können durch Definition der Umgebungsvariablen CMXTRACE und/oder NEATRACE ein- bzw. ausgeschaltet werden:
  CMXTRACE=-io; export CMXTRACE und/oder
  NEATRACE=-io[...weitere Schalter..]; export NEATRACE.

  Das Ein-/Ausschalten wird zum Zeitpunkt des x_attach Aufrufes (wie bei CMX) wirksam.

  Die Verfolgerinformation wird in eine Datei mit dem Namen nea.trc.<pid> im gerade genutzten Directory abgelegt und ist druckaufbereitet.
  Der Ausdruck erfolgt mit lpr -pb3/-pb2 nea.trc.<pid>.
  Sie enthält Datum und Uhrzeitangaben. Die NEABX-Aufrufe tragen einen Zeitstempel, der sie dem CMX-Aufruf zuordnen läßt.

- Weitere Einstellungen der Umgebungsvariablen:
  Trace aus: NEATRACE=-, ohne nachfolgenden 'io'.
  Ausdruck größerer Datenlängen: L (max. 4096), Ln (max. n).
  Ausdruck aller relevanten Parameter: p
  Ausdruck von Daten auch in abdruckbarer Form: a
  Ausdruck detaillierter Beschreibung weiterer Schalter im Trace: u

Ausdruck von Zeitangaben im Trace

Am Kopf des Trace wird Datum und Uhrzeit ausgedruckt.
Beim Verlassen jedes x_Aufrufes wird die exakte Tageszeit (sekundengenau) ausgedruckt, wodurch eine zeitliche Zuordnung zu CMX-Trace möglich ist.

```
Trace File für die NEABX - Anwendung mit pid 899:

Datum/Uhrzeit 24.09.87,  12:22:01

Tracevariable: 'CMXTRACE= nicht definiert'; 'NEATRACE=ioapuL200';

Kurzbeschreibung d. Trace-Schalter:
  Trace-Ein: NEATRACE=-io oder CMXTRACE=-io;
  ausfuehrl. Schalterbeschrbg.: u;
  Groessere Datenlng: L (max.4096), L[n] (max.n);
  Weitere Sch.: a,p,X[n],T[n];
Beispiel: NEATRACE=iopaL200;

Beschreibung v. NEATRACE=[..schalter..]:
-  NEA-Trace-Ein: io; NEA-TRACE-Aus: '-' (ohne folgendem i/o);
-  a: Daten hex und abdruckbar; Druck 96-spaltig !!!
-  p: Tr. d. relev. Aufruf- bzw. Return-Par.;
-  Daten-Tr.-Lng.: L[n], X[n], T[n];
   L-Angaben: Begrenzung der Tracelng. von neabx-Aufruf-Daten,
      wenn keine speziell. X[n]/T[n]-Angaben gemacht wurden.
      kein L-Ang.: max=20; L: max.=4096; Ln: max.=n;
   X, X[n], keine X-Angabe: Begrenzg. NEABX-Daten-Trace:
      kein X-Ang.: max=20; X: max.=4096; Xn: max.=n;
   T, T[n], keine T-Ang.: Begrenzg. CMX-Daten-Trace:
      keine T-Ang.: max=L-Angabe; T: max.=4096; Tn: max.=n;

Beispiel: NEATRACE=-iouapX40T1080L100;
  x_.. Aufrufdaten in max. Lng.=40, t_.. max. Lng.=1080 u.
  andere Daten in max. Lng.=100;

Aufruf-| Nr. in | NEABX| ausgwaehlte od. detaillierte  | CMX- | Daten:
Zaehler| x_Funk.| Funk.| Parameter bei Aufruf/Rueckgabe| Aufr.|

001> 01. x_attach, BEGIN, name: 0x2bc8c, x_opt: 0x0
         (x_my)name-Par  x_mnmode: 0x01, x_mnres: 0x00, x_mnlng: 24, x_mn: 0x..
         0e 00 ic 00 00 00 04 00 0a 00 45 4d 44 53 53 30 30 30 00 00   | ..........EMDSS000..
         00 00 00 00                                                   | ....
001> 02. x_attach, EMD, name: 0x2bc8c, ret:1
------------------------------------------------------- TIME : ** 12:22:03 [hh:mm:ss] **

002> 01. x_detach, BEGIN, name: 0x2bc8c, x_opt: 0x0
         (x_my)name-Par  x_mnmode: 0x01, x_mnres: 0x00, x_mnlng: 24, x_mn: 0x..
         0e 00 1c 00 00 00 04 00 0a 00 45 4d 44 53 53 30 30 30 00 00   | ..........EMDSS000..
         00 00 00 00                                                   | ....
002> 02. x_detach, EMD, name: 0x2bc8c, ret:0
------------------------------------------------------- TIME : ** 12:22:03 [hh:mm:ss] **

003> 01. x_attach, BEGIN, name: 0x2bc8c, x_opt: 0x0
         (x_my)name-Par  x_mnmode: 0x01, x_mnres: 0x00, x_mnlng: 24, x_mn: 0x..
         0e 00 12 00 00 00 04 00 0a 00 45 4d 44 53 53 30 30 31 00 00   | ..........EMDSS001..
         00 00 00 00                                                   | ....
003> 02. x_attach, EMD, name: 0x2bc8c, ret:1
------------------------------------------------------- TIME : ** 12:22:06 [hh:mm:ss] **

004> 01. x_detach, BEGIN, name: 0x2bc8c, x_opt: 0x0
         (x_my)name-Par  x_mnmode: 0x01, x_mnres: 0x00, x_mnlng: 24, x_mn: 0x..
         0e 00 12 00 00 00 04 00 0a 00 45 4d 44 53 53 30 30 31 00 00   | ..........EMDSS001..
         00 00 00                                                      | ....
004> 02. x_detach, EMD, name: 0x2bc8c, ret:0
------------------------------------------------------- TIME : ** 12:22:06 [hh:mm:ss] **

005> 01. x_attach, BEGIN, name: 0x2bc8c, x_opt: 0x0
         (x_my)name-Par  x_mnmode: 0x01, x_mnres: 0x00, x_mnlng: 24, x_mn: 0x..
         0e 00 13 00 00 00 04 00 0a 00 45 4d 44 53 53 30 30 32 00 00   | ..........EMDSS002..
         00 00 00 00                                                   | ....
005> 02. x_attach, EMD, name: 0x2bc8c, ret:0
------------------------------------------------------- TIMER : ** 12:22:06 [hh:mm:ss] **
```

A Anhang

```
/*
 * @(#)cmx.h    2.18 87/07/30 CMX (DF1)
 * $Header: $
 */

/*
 * COPYRIGHT (C) 1987 BY SIEMENS AG
 * All rights reserved
 *
 * This software is furnished under a license and may be used
 * only in accordance with the terms of that license and with the
 * inclusion of the above copyright notice. This software may not
 * be provided or otherwise made available to, or used by, any
 * other person. No title to or ownership of the software is
 * hereby transferred.
 */

/*****************************************************************/
/*                                                               */
/*                      HEADER-FILE  (cmx.h)                     */
/*                                                               */
/*    fuer die Library Schnittstelle zum Transportsystem in Sinix    */
/*                                                               */
/*                          I C M X (L)                          */
/*                                                               */
/*  Datum der letzten Aenderung: 27.07.87          Version: V2.1   */
/*****************************************************************/

/*****************************************************************/
/*                          KONSTANTEN                           */
/*****************************************************************/

#define T_OK            0       /* Funktionsaufruf erfolgreich */
#define T_ERROR        -1       /* Funktionsaufruf fehlerhaft */
#define T_NOTFIRST      1       /* Returnwert bei t_attach */
#define T_DATASTOP     -2       /* Returnwert bei t_datarq */
#define T_XDATSTOP     -3       /* Returnwert bei t_xdatrq */

/*
 * Returnwerte bei t_event
 */
#define T_NOEVENT       0       /* kein Ereignis vorhanden */
#define T_DATAIN        1       /* Datenempfangsanzeige */
#define T_DATAGO        2       /* Daten wieder sendbar */
#define T_XDATIN        3       /* Datenempfangsanzeige fuer Vorrangdaten */
#define T_XDATGO        4       /* Vorrangdaten wieder sendbar */
#define T_CONIN         5       /* Verbindungsaufbau-Anzeige */
#define T_CONCF         6       /* Verbindungsaufbau-Bestaetigung */
#define T_DISIN         7       /* Verbindungsabbau-Anzeige */
#define T_REDIN         8       /* Verbindungsumlenkungs-Anzeige */

/*
 * Optionsnummern
 */
#define T_OPTA1         1       /* Options-Nr. fuer struct t_opta1 */
#define T_OPTC1         1       /* Options-Nr. fuer struct t_optc1 */
#define T_OPTC2         2       /* Options-Nr. fuer struct t_optc2 */
#define T_OPTI1         1       /* Options-Nr. fuer struct t_opti1 */

/*
 * Werte fuer Parameter t_apmode
 */
#define T_ACTIVE        1       /* nur aktiver Verbindungsaufbau */
#define T_PASSIVE       2       /* nur passiver Verbindungsaufbau */
#define T_REDIRECT      4       /* nur Verbindungsumlenkungen */

/*
 * Werte fuer Parameter cmode
 */
#define T_WAIT          0       /* Warte synchron auf ein Ereignis */
#define T_CHECK         1       /* Pruefe, ob Ereignis da ist */

/*
 * Werte fuer Parameter t_xdata, t_timeout
 */
#define T_YES           1       /* Vorrangdaten erwuenscht */
```

```
#define T_NO            0       /* keine Vorrangdaten oder keine Ueberwachung */

/*
 * Werte fuer Parameter reason (Grund des Verbindungsabbaus)
 *
 * 1. Allgemein verwendet
 */
#define T_USER          0       /* Abbau erfolgt durch Partner (Benutzer),
                                   u.U. auch Benutzerfehler des Partners */
/*
 * 2. Bei Rechnerkopplung verwendet
 */
#define T_RTIMEOUT      1       /* Abbau lokal durch CMX wegen Inaktivitaet
                                   der Verbindung gemaess Parameter t_timeout */
#define T_RADMIN        2       /* Abbau lokal durch CMX wegen Ausserbetrieb-
                                   nahme des CCP durch die Administration */
#define T_RCCPEND       3       /* Abbau lokal durch CMX wegen CCP-Ausfall */
#define T_RCCP        256       /* bei allen reasons >= T_RCCP ist der Verb.-
                                   abbau vom CCP veranlasst; alle diese reasons
                                   haben die Form T_RCCP + ITRANS-reason */
#define T_RUNKNOWN    256       /* Abbau vom Partner-CCP, Grund nicht angegeben,
                                   u.U. auch Benutzerfehler des Partners */
#define T_RSAPCONGEST 257       /* Abbau vom Partner-CCP wegen T-SAP-spezif.
                                   Engpass */
#define T_RSAPNOTATT  258       /* Abbau vom Partner-CCP, weil der adressierte
                                   T-SAP dort nicht angemeldet ist */
#define T_RUNSAP      259       /* Abbau vom Partner-CCP, weil der adressierte
                                   T-SAP dort nicht bekannt ist */
#define T_RCONGEST    385       /* Abbau vom Partner-CCP wegen Betriebsmittel-
                                   engpass */
#define T_RCONNFAIL   386       /* Abbau vom Partner-CCP wegen Misslingen des
                                   Verbindungsaufbaus, z.B. weil Benutzerdaten
                                   zu lang */
#define T_RDUPREF     387       /* Abbau vom Partner-CCP, weil fuer ein N-SAP-
                                   Paar eine zweite Verbindungsreferenz ver-
                                   geben wurde (Systemfehler) */
#define T_RMISREF     388       /* Abbau vom Partner-CCP wegen einer nicht
                                   zuzuordnenden Verbindungsreferenz
                                   (Systemfehler) */
#define T_RPROTERR    389       /* Abbau vom Partner-CCP wegen Transport-
                                   protokollfehler, z.B. Vorrangdaten erhalten,
                                   obwohl nicht erlaubt */
#define T_RREFOFLOW   391       /* Abbau vom Partner-CCP wegen Verbindungs-
                                   referenz-Ueberlauf */
#define T_RNOCONN     392       /* Aufbau der Netzverbindung vom Partner-CCP
                                   abgelehnt */
#define T_RINLNG      394       /* Abbau vom Partner-CCP wegen falscher Header-
                                   oder Parameterlaenge (Systemfehler) */
#define T_RLCONGEST   448       /* Abbau vom lokalen CCP wegen Betriebsmittel-
                                   engpass */
#define T_RLNOQOS     449       /* Abbau vom lokalen CCP, weil Quality of
                                   Service nicht mehr geboten werden kann */
#define T_RLPROTERR   464       /* Abbau vom lokalen CCP wegen Transport-
                                   protokollfehler, z.B. Vorrangdaten erhalten,
                                   obwohl nicht erlaubt */
#define T_RLINTIDU    465       /* Abbau vom lokalen CCP, weil es eine zu
                                   lange Schnittstellen-Dateneinheit (TIDU)
                                   erhalten hat (Systemfehler) */
#define T_RLNORMFLOW  466       /* Abbau vom lokalen CCP wegen Verletzung der
                                   Flussregelungsvorschrift fuer Normaldaten
                                   (Systemfehler) */
#define T_RLEXFLOW    467       /* Abbau vom lokalen CCP wegen Verletzung der
                                   Flussregelungsvorschrift fuer Vorrangdaten
                                   (Systemfehler) */
#define T_RLINSAPID   468       /* Abbau vom lokalen CCP, weil es eine un-
                                   gueltige T-SAP-Id. erhalten hat (Systemf.) */
#define T_RLINCEPID   469       /* Abbau vom lokalen CCP, weil es eine un-
                                   gueltige T-CEP-Id. erhalten hat (Systemf.) */
#define T_RLINPAR     470       /* Abbau vom lokalen CCP wegen eines unzu-
                                   laessigen Parameterwerts, z.B. Benutzerdaten
                                   oder Vorrangdaten zu lang */
#define T_RLNOPERM    480       /* Aufbau durch Administration des lokalen CCP
                                   verhindert */
#define T_RLPERMLOST  481       /* Abbau durch Administration des lokalen CCP */
#define T_RLNOCONN    482       /* Aufbau vom lokalen CCP nicht durchfuehrbar,
                                   weil keine Netzverbindung zustande kommt */
#define T_RLCONNLOST  483       /* Abbau vom lokalen CCP wegen Verlust der
                                   Netzverbindung */
#define T_RLNORESP    484       /* Aufbau vom lokalen CCP nicht durchfuehrbar,
                                   weil Partner nicht auf CONRQ antwortet */
#define T_RLIDLETRAF  485       /* Abbau vom lokalen CCP wegen Verlust der Ver-
                                   bindung (Idle Traffic Inactivity Timeout) */
```

```
#define T_RLRESYNC      486     /* Abbau vom lokalen CCP, weil Resynchroni-
                                   sierung erfolglos war (mehr als 10 Wdh.) */
#define T_RLEXLOST      487     /* Abbau vom lokalen CCP, weil Vorrangdaten-
                                   kanal defekt ist (mehr als 3 Wdh.) */
/*
 * 3. Bei Stationskopplung und in V1.0 verwendet
 */
#define T_SYS4            4     /* Partner antwortet nicht innerhalb des
                                   Zeitlimits */
#define T_SYS8            8     /* Partner nicht verfuegbar */
#define T_SYS12          12     /* Verbindungsparameter nicht akzeptiert */
#define T_SYS20          20     /* Betriebsmittelengpass */
#define T_SYS28          28     /* Systemabschaltung */
#define T_SYS32          32     /* Systemfehler */
#define T_SYS56          56     /* Abbauanforderung von Systemadministration */
#define T_SYS64          64     /* Partner unbekannt */
#define T_SYS255        255     /* Systemfehler */

/*
 * Werte fuer Parameter chain
 */
#define T_MORE            1     /* weitere Daten in der TSDU */
#define T_END             0     /* keine weiteren Daten in der TSDU */

/*
 * Laengendefinitionen
 */
#define T_MSG_SIZE      109     /* Max. Nachrichtenlng. beim Verb.-aufbau */
#define T_RED_SIZE       32     /* Max. Nachrichtlng. bei Verb.-umlenkung */
#define T_EXP_SIZE       16     /* Max. Nachrichtenlaenge bei Vorrangdaten */

/*
 * Fehlerinformationen
 * die von der Funktion t_error() zurueckgelieferte Fehlerinformation ist
 * folgendermassen strukturiert:
 *      Bit 2**0  - 2**7  = Fehlerwert
 *      Bit 2**8  - 2**11 = Fehlerklasse
 *      Bit 2**12 - 2**15 = Fehlertyp
 */

/*
 * Fehlertypen
 */
#define T_CMXTYPE       (0 << 12)       /* CMX-Fehler */
#define T_DSTEMP_ERR    (2 << 12)       /* TNSX-Fehler (temporaerer Fehler) */
#define T_DSCALL_ERR    (3 << 12)       /*      "      (Aufruffehler) */
#define T_DSPERM_ERR    (4 << 12)       /*      "      (permanenter Fehler) */
#define T_DSWARNING     (5 << 12)       /*      "      (Warnung) */
#define T_BX1           (7 << 12)       /* NEABX-Fehler */
#define T_BX2           (8 << 12)       /* NEABX-Fehler */
#define T_BX3           (9 << 12)       /* NEABX-Fehler */

/*
 * Fehlerklassen fuer CMX und TNSX
 */
#define T_CMXCLASS      (0 << 8)        /* CMX-Fehler */
#define T_DSNOT_SPEC    (2 << 8)        /* TNSX-Fehler (nicht naeher spezifizierter Fehler) */
#define T_DSPAR_ERR     (3 << 8)        /*      "      (Parameterfehler) */
#define T_DSILL_VERS    (4 << 8)        /*      "      (ungueltige Version) */
#define T_DSSYS_ERR     (5 << 8)        /*      "      (Systemfehler) */
#define T_DSINT_ERR     (6 << 8)        /*      "      (interner Fehler) */
#define T_DSMESSAGE     (7 << 8)        /*      "      (Hinweis) */

/*
 * Fehlerklassen fuer NEABX: Siehe "neabx.h"
 */

/*
 * Werte der TNSX-Fehler: Siehe "tnsx.h"
 *
 * Werte der NEABX-Fehler: Siehe "neabx.h"
 *
 * Werte der CMX-Fehler: Es gibt drei Gruppen von CMX-Fehlern, naemlich
 * - allgemeine CMX-Fehler
 * - zusaetzliche Fehler bei Stationskopplung ueber MSV1
 * - zusaetzliche Fehler bei Stationskopplung ueber BAM
 * Die Werte sind i.f. definiert.
 */

/*
 * Allgemeine CMX-Fehler
 */
```

```
#define T_NOERROR         0      /* kein Fehler */
#define T_ENOENT          2      /* kein Speicher fuer weitere Verbindung vorh.,
                                    oder zu viele Prozesse haben Anwendungen
                                    angemeldet */
#define T_EIO             5      /* CCP-Ausfall */
#define T_EBADF           9      /* Funktionsaufruf in diesem Zustand nicht
                                    zulaessig */
#define T_ENOMEM         12      /* kein Speicher fuer weitere
                                    Verbindungsumlenkung vorhanden */
#define T_EFAULT         14      /* Fehler beim Kopieren von/nach User-Bereich */
#define T_EBUSY          16      /* Netz-Geraetedatei bereits geoeffnet (diese
                                    Meldung erscheint nicht an der ICMX-L) */
#define T_EINVAL         22      /* fehlerhafter Parameter */
#define T_ENOBUF         37      /* temporaerer Betriebsmittelengpass */
#define T_UNSPECIFIED   100      /* nicht naeher spezifierter Fehler */
#define T_WSEQUENCE     101      /* Funktionsaufruf in diesem Zustand nicht
                                    zulaessig */
#define T_WREQUEST      102      /* unzulaessiger Funktionsaufruf nach fork */
#define T_WPARAMETER    103      /* fehlerhafter Parameter */
#define T_WAPPLICATION  104      /* Anwendung von diesem Prozess noch nicht
                                    angemeldet (t_conrq, t_detach) bzw. schon
                                    angemeldet (t_attach) */
#define T_WAPP_LIMIT    105      /* zu viele Anwendungen oder Anmeldungen
                                    gleichzeitig */
#define T_WCONN_LIMIT   106      /* zu viele Verbindungen gleichzeitig */
#define T_WTREF         107      /* unzulaessige Transportreferenz */
#define T_WTUI_MSG      108      /* Fehlerhafte TUI-Netzmeldung
                                    (nur bei Stationskopplung) */
#define T_COLLISION     109      /* Kollision aktiver Verbindungsaufbau /
                                    passiver Verbindungsaufbau
                                    (bzw. Verbindungsumlenkung) oder
                                    Kollision Aufruf mit bereits vorliegendem
                                    T_DISIN-Ereignis */
#define T_WPROC_LIMIT   110      /* zu viele Prozesse haben Anwendungen
                                    angemeldet */
#define T_NOCCP         111      /* Kein CCP fuer gewuenschte Anwendung oder
                                    Verbindung vorhanden */
#define T_ETIMEOUT      112      /* CCP reagiert nicht rechtzeitig */
#define T_WROUTINFO     113      /* Unzulaessige Routing Information (CC-Index)
                                    beim Verbindungsaufbau */
#define T_CCP_END       114      /* CCP ist nicht mehr betriebsbereit */
#define T_WRED_LIMIT    115      /* zu viele Verbindungsumlenkungen gleichztg.*/
#define T_WLIBVERSION   116      /* Version der verwendeten CMX-Bibliothek
                                    passt nicht zum CMX-Treiber */

/*
 * Fehlerwerte bei Stationskopplung ueber CCP-STA1
 */
#define T_EPARUOPEN      48      /* Ein Kanal ist eroeffnet, keine
                                    Parametrisierung moeglich */
#define T_EFPARAM        49      /* Parameter-Fehler, Standleitung, halbduplex
                                    nicht zulaessig */
#define T_EPARAM         50      /* keine Parameter-Werte vorhanden */
#define T_EMODFL         51      /* Modem-Fehler, M1 ohne S1 gesetzt */
#define T_ECTS           52      /* CTS/M2 nicht gesetzt innerhalb 2 sec,
                                    Meldung erfolgt alle 2 sec */
#define T_EDSR           53      /* DSR/M1 nicht mehr gesetzt, Prozedurabbruch */
#define T_EDLEEOT        54      /* Prozedurabbruch, DLE EOT empfangen,
                                    nur bei einer Waehlleitung */
#define T_EWTOUT24       55      /* kein Steuerzeichen-Empfang innerhalb 24 s
                                    bei Waehlleitung, Prozedurabbruch */
#define T_ESTOUT24       56      /* kein Steuerzeichen-Empfang innerhalb 24 s
                                    bei Standleitung */
#define T_EWRCOUNT       57      /* Anzahl der auszusendenden Daten zu gross */
#define T_EWRNGC         58      /* auszusendende Daten enthalten Prozedur
                                    Steuerzeichen */
#define T_EPRRWZ         64      /* MSV1-Prozedurfehler beim Empfang, Text falsch
                                    empfangen, Wiederholzaehler abgelaufen */
#define T_EPRREOT        65      /* MSV1-Prozedurfehler beim Empfang, unerwar-
                                    tetes EOT nach STX Empfang */
#define T_EPRXWZ         66      /* MSV1-Prozedurfehler beim Senden, Text konnte
                                    nicht gesendet werden, Wiederhohlzaehler
                                    abgelaufen */
#define T_EPRXWABT       67      /* MSV1-Prozedurfehler beim Senden, Text gut
                                    gesendet, aber WABT-Zaehler abgelaufen */
#define T_EPROZEMPF      80      /* jetzt wieder Prozedur-Zeichen empfangen */

/*
 * Fehlerwerte bei Stationskopplung ueber CCP-STA2
 */
#define T_EPARUOPEN      48      /* DSSPAR-ioctl nicht zulaessig, wenn schon
                                    ein device offen ist */
#define T_EPARAM         50      /* keine Leitungsparameter vorhanden */
```

```
#define T_EMODFL        51      /* Open: (HW)-Error */
#define T_ECTS          52      /* Modem oder HW-Fehler*/
#define T_EDSR          53      /* Prozedurabbruch (Ltg nicht mehr vorhanden?)*/
#define T_ESTOUT24      56      /* Polling ausgefallen */
#define T_EWRCOUNT      57      /* Anzahl der Sende-/Empfangsdaten falsch */
#define T_EPRRWZ        64      /* Daten konnten nicht empfangen werden */
#define T_EPRXWZ        66      /* Daten konnten nicht gesendet werden */
#define T_EDATLST       68      /* Datenverlust beim Empfang */
#define T_EPROZEMPF     80      /* Polling wieder registriert */

/*********************************************************************/
/*                          STRUKTUREN                               */
/*********************************************************************/

struct t_opta1 {
        int t_optnr;            /* Options-Nr. */
        int t_apmode;           /* Anwendungs-Mode */
        int t_conlim;           /* max. Anzahl der Verbindungen */
};

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

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

struct t_opti1 {
        int t_optnr;            /* Options-Nr. */
        int t_maxl;             /* Laenge der TIDU */
};

/*********************************************************************/
/*                          NAMEN UND ADRESSEN                       */
/*********************************************************************/

/*
 * Nur in CMX V1.0 unterstuetzte Adressformate (kein TNSX)
 */
#define T_TRANSDATA     '0'

struct t_tdaddr {
        char t_admode;          /* = T_TRANSDATA (Adressierungs-Modus) */
        char t_pname[8];        /* TRANSDATA-Rechner-/Regionsnummer */
        char t_stname[8];       /* TRANSDATA-Stationsname */
        char t_term;            /* = \0 (Endekennzeichen) */
};

#define T_T70           '1'

struct t_t70addr {
        char t_admode;          /* = T_T70 (Adressierungs-Modus) */
        char t_netadr[21];      /* Netz-Wahlinformation */
        char t_tsapid[21];      /* TSAP-Identifier */
        char t_protocolid[5];   /* X.25 'protocolid' */
};

/*
 * Ab CMX V2.0 unterstuetzte Adressformate im Zusammenhang mit TNSX
 */
#define T_MNMODE 1
#define T_MNSIZE 120

struct t_myname {
        char t_mnmode;          /* = T_MNMODE */
        char t_mnres;           /* = 0 */
        short t_mnlng;          /* Laenge des ausgefuellten Teils der Struktur*/
        char t_mn[T_MNSIZE];    /* Feld fuer die T-Selektoren */
};

#define T_PAMODE 2
#define T_PASIZE 108

struct t_partaddr {
        char t_pamode;          /* = T_PAMODE */
        char t_pares;           /* = 0 */
```

```
        short t_palng;          /* Laenge des ausgefuellten Teils der Struktur*/
        char t_pa[T_PASIZE];    /* Feld fuer die Partneradresse */
};

union t_address {
        struct t_myname tmyname;
        struct t_partaddr tpartaddr;
        struct t_tdaddr tdaddr;
        struct t_t70addr t70addr;
};

/*********************************************************************/
/*             DEFINITION DER ICMX(L)-AUFRUFE                        */
/*********************************************************************/

int     t_attach();
int     t_detach();
int     t_event();
int     t_conrq();
int     t_conin();
int     t_conrs();
int     t_concf();
int     t_disrq();
int     t_disin();
int     t_redrq();
int     t_redin();
int     t_datarq();
int     t_datain();
int     t_datastop();
int     t_datago();
int     t_xdatrq();
int     t_xdatin();
int     t_xdatstop();
int     t_xdatgo();
int     t_info();
int     t_error();
```

```
/*
 * COPYRIGHT (C) 1987 BY SIEMENS AG
 * All rights reserved
 *
 * This software is furnished under a license and may be used
 * only in accordance with the terms of that license and with the
 * inclusion of the above copyright notice. This software may not
 * be provided or otherwise made available to, or used by, any
 * other person. No title to or ownership of the software is
 * hereby transferred.
 */

/*****************************************************************************/
/*                                                                           */
/*        HEADER-FILE  FUER "TRANSPORT NAME SERVICE"-SCHNITTSTELLE           */
/*                                                                           */
/*  @(#)tnsx.h  2.9 87/07/17 TNSX (DF 1)                                     */
/*                                                                           */
/* ------------------------------------------------------------------------- */
/*  Autor: R. Horn                                  Datum: 10.2.86           */
/*****************************************************************************/

/* ***************** DEFINITION VON KONSTANTEN ************************** */

        /* Versionsnummern */
#define TS_1VER01               1       /* Versionsnummer fuer Funktionen zur
                                           Informationsabfrage */
#define TS_2VER01               1       /* Versionsnummer fuer Funktionen zur
                                           Informationsverwaltung */

        /* Systemparameter */
#define TS_NO_NP                5       /* max. Anzahl der Namensteile */
#define TS_MAX_NO_OF_PROPERTIES 20      /* max. Anzahl der Eigenschaften, die */
                                        /* einem Objekt zugeordnet werden koennen */
#define TS_MX_X_USERS           500     /* max. Anzahl der verwaltbaren Objekte */
                                        /* fuer PC-MX bzw. PC-X */
#define TS_MX2_MX4_USERS        500     /* max. Anzahl der verwaltbaren Objekte */
                                        /* fuer PC-MX2 bzw. PC-MX4 */
#define TS_CMX_DIR_ID           1       /* Identifier fuer CMX-Directory */

        /* Werte fuer den Typ von Namensteilen */
#define TS_COUNTRY              66000   /* Country */
#define TS_ADMD                 66001   /* Administration domain */
#define TS_PRMD                 66002   /* Private domain */
#define TS_OU                   66003   /* Organisation unit */
#define TS_PN                   66004   /* Personal name */

        /* Werte fuer die maximale Laenge von Namensteilen */
#define TS_LCOUNTRY             2       /* Country */
#define TS_LADMD                16      /* Administration domain */
#define TS_LPRMD                16      /* Private domain */
#define TS_LOU                  10      /* Organisation unit */
#define TS_LPN                  30      /* Personal name */

        /* Wert fuer die maximale Laenge des Wertes einer Eigenschaft */
#define TS_LPROP                200

        /* Werte fuer den Namen von Eigenschaften ("property-id") */
#define TS_EMPTYPROP            0       /* Endekennung einer Property-Liste oder
                                           Kennzeichen fuer fehlenden Filter */
#define TS_TRANS                100     /* Transportadresse */
#define TS_NEABX                102     /* Verbindungsmodus (Anwendung mit/
                                           ohne NEABX-Service) */
#define TS_ROUT                 103     /* Routinginformation (Board-, Port-
                                           nummer */
#define TS_GTYPE                105     /* Geraeteprotokoll (z.B. 810) */
#define TS_TRSYS                106     /* Transportsystem (NEA, ISO, LAN, STA) */
#define TS_LNAME                107     /* lokale Namen einer Anwendung */
                                        /* (T-Selektoren) */
#define TS_USER1PR              120     /* reservierte Werte fuer be- */
#define TS_USER2PR              121     /* nutzerspezifische Eigenschaften */
#define TS_USER3PR              122

        /* Werte fuer den Typ von Eigenschaften ("property-type") */
#define TS_ITEM                 0
#define TS_GROUP                1       /* Gruppen-Eigenschaft
                                           (nicht unterstuetzt in Version 1) */

        /* Funktionsrueckgabewerte */
#define TS_OK                   0       /* Funktion erfolgreich abgeschlossen */
#define TS_ERROR                -1      /* Funktion mit Fehler abgeschlossen */
```

```
        /*FEHLERWERTE: bei Funktionsrueckgabewert "TS_ERROR" */
        /* Fehlerwerte fuer "ts_retcode" */
#define TS_NOERR                0       /* kein Fehler */
#define TS_TEMPERR              -1      /* temporaerer Fehler */
#define TS_CALLERR              -2      /* Aufruffehler */
#define TS_PERMERR              -3      /* permanenter Fehler */
#define TS_WARNING              -99     /* Warnung */

        /* Fehlerwerte fuer "ts_errclass" */
#define TS_NOTSPEC              0       /* nicht naeher spezifizierter Fehler */
                                        /* Wert kann auch bei "ts_errvalue" auftreten */
#define TS_PARERR               1       /* Parameterfehler */
#define TS_ILLVERS              3       /* ungueltige Version */
#define TS_SYSERR               4       /* Systemfehler */
#define TS_INTERR               6       /* interner Fehler */
#define TS_MESSAGE              99      /* Hinweis */

        /* Fehlerwerte fuer "ts_errvalue" */
/* Fehlerwerte sind in Zusammenhang mit den Wert fuer "ts_errclass" zu betrachten */
/* ---- im Fall ts_errclass = TS_PARERR */

#define TS_DIRERR               1       /* Directory unbekannt */
#define TS_NAMERR               2       /* der angegebene Name ist nicht vor- */
                                        /* handen bzw. existiert bereits (funktions- */
                                        /* abhaengige Bedeutung) */
#define TS_ILLNAM               3       /* Name syntaktisch falsch (z.B. Wert eines */
                                        /* Namensteils zu lang, falscher Identifier, */
                                        /* zu viele Namensteile) */
#define TS_STRERR               4       /* unzulaessiger Wert fuer "ts11str_def" */
#define TS_PROPER               5       /* unbekannte Eigenschaft oder */
                                        /* keine Eigenschaft angegeben oder */
                                        /* Eigenschaft existiert bereits */
#define TS_SCOPER               6       /* unzulaessiger Wert fuer "ts11scope" */
#define TS_MEMORY               9       /* zur Verfuegung gestellter Speicher- */
                                        /* platz fuer die Werte von z.B. Namens- */
                                        /* teilen oder Eigenschaften nicht ausreichend */
#define TS_SIDERR               12      /* unbekanntes Kurzkennzeichen */
#define TS_ILLUSE               13      /* Verwendung eines Kurzkennzeichens nicht erlaubt */
#define TS_OBJXST               15      /* der angegebene Objektname existiert bereits */
#define TS_LENERR               16      /* unzulaessige Laenge des Wertes einer Eigenschaft */
                                        /* oder des Pfadnamens der Ausgabedatei */

/* ---- im Fall ts_errclass = TS_ILLVERS */

/*#define TS_NOTSPEC            0 */

/* ---- im Fall ts_errclass = TS_SYSERR */

/* Fehlerwerte siehe SINIX-Systemmanual */

/* ---- im Fall ts_errclass = TS_INTERR */

#define TS_TIMOUT               7       /* kein DSA vorhanden */
/*      TS_MEMORY               9          der vom System angeforderte Speicher
                                           fuer den Austausch von Protokollnach-k
                                           richten zwischen DUA < > DSA steht nicht
                                           zur Verfuegung bzw. eine Protokoll-
                                           nachricht ist groesser als der da-
                                           fuer reservierte Speicherbereich (bei
                                           den Funktionen "TS_ReadLeafs" bzw.
                                           "TS_ReadChildren") */
#define TS_UPDERR               14      /* Directory voll (overflow) */
#define TS_PROT                 20      /* Protokollfehler zwischen DUA < > DSA */
#define TS_NORQ                 21      /* DSA kann momentan keine Anforderung bearbeiten */
#define TS_LFILE                100     /* fehlerhafte Laenge einer Directory-Datei */

/* ---- im Fall ts_errclass = TS_MESSAGE */

#define TS_TIMLIM               8       /* Funktionsausfuehrung abgebrochen wegen
                                           Erreichens der Zeitschranke */
#define TS_LEAFNO               10      /* Anzahl der gefundenen Objektnamen
                                           groesser als die der erwarteten */
#define TS_CHLDNO               11      /* Anzahl der gefundenen "children"-Objektnamen
                                           groesser als die der erwarteten */


/* ***************** DEFINITION VON DATENTYPEN ************************** */

        /* Basisdatentypen */

typedef char Ts_Bool ;
#define TS_FALSE        0
#define TS_TRUE         1
```

```
typedef short Ts_str_def ;
#define TS_RESTRICTED   0
#define TS_GENERAL      1

typedef short Ts_scope ;
#define TS_LOCAL        0
#define TS_GLOBAL       1

        /* Struktur des Standardkopfes */
typedef struct {
        short           ts_version ;
        short           ts_retcode ;
        short           ts_errclass ;
        short           ts_errvalue ;
        } Ts_head ;

        /* Struktur einer Eigenschaft */
typedef struct {
        short           ts_name ;
        short           ts_type ;
        short           ts_length ;
        char            *ts_value ;
        } Ts_property ;

        /* Struktur einer als Filter verwendeten Eigenschaft */
typedef struct {
        short           ts_pr_name ;
        short           ts_pr_length ;
        char            *ts_pr_value ;
        } Ts_filter ;

        /* Struktur eines Namensteils */
typedef struct {
        long            ts_np_id ;
        short           ts_np_length ;
        char            *ts_np_value ;
        } Ts_name_part ;

        /* Struktur eines Namens (mit Eigenschaften) */
typedef struct {
        short           ts_no_of_el ;
        Ts_name_part    ts_lf[TS_NO_NP] ;
        } Ts_leaf_name ;

        /* Struktur eines Namens (ohne Eigenschaften) */
typedef struct {
        short           ts_no_of_el ;
        Ts_name_part    ts_nlf[TS_NO_NP - 1] ;
        } Ts_non_leaf_name ;

        /* Struktur des Namens einer Parameter-Datei */
typedef struct {
        char            *ts_path ;
        } Ts_pfname ;

        /* Parameterstruktur fuer Funktion "TS_ReadLeafs" */
typedef struct {
        Ts_head         *ts_head ;
        short           ts11dir_id ;
        Ts_leaf_name    *ts11partial_name ;
        Ts_str_def      ts11str_def ;
        Ts_filter       *ts11filter ;
        Ts_scope        ts11scope ;
        Ts_pfname       ts11filename ;
        short           ts11time_limit ;
        char            *ts11npval ;
        short           ts11l_np_values ;
        short           ts11exp_no ;
        Ts_leaf_name    *ts11leafs ;
        short           ts11real_no ;
        } Ts_P11 ;

        /* Parameterstruktur fuer Funktion "TS_ReadChildren" */
typedef struct {
        Ts_head         *ts_head ;
        short           ts12dir_id ;
        Ts_non_leaf_name *ts12parent ;
        Ts_filter       *ts12filter ;
        char            *ts12npval ;
        short           ts12l_np_values ;
        short           ts12exp_no ;
        Ts_name_part    *ts12children ;
```

```
        short           ts12real_no ;
        } Ts_P12 ;

        /* Parameterstruktur fuer Funktion "TS_ReadProperties" */
typedef struct {
        Ts_head         *ts_head ;
        short           ts13dir_id ;
        Ts_leaf_name    *ts13leaf_name ;
        long            ts13short_id ;
        char            *ts13prval ;
        short           ts13l_pr_values ;
        Ts_property     *ts13prop ;
        Ts_leaf_name    *ts13dist_name ;
        char            *ts13npval ;
        short           ts13l_np_values ;
        } Ts_P13 ;

        /* Parameterstruktur fuer Funktion "TS_CreateNonLeafEntity" */
typedef struct {
        Ts_head          *ts_head ;
        short            ts21dir_id ;
        Ts_non_leaf_name *ts21parent ;
        } Ts_P21 ;

        /* Parameterstruktur fuer Funktion "TS_DeleteNonLeafEntity" */
typedef struct {
        Ts_head          *ts_head ;
        short            ts22dir_id ;
        Ts_non_leaf_name *ts22parent ;
        } Ts_P22 ;

        /* Parameterstruktur fuer Funktion "TS_CreateLeafEntity" */
typedef struct {
        Ts_head          *ts_head ;
        short            ts23dir_id ;
        Ts_non_leaf_name *ts23parent ;
        Ts_name_part     *ts23dist_arc ;
        Ts_property      *ts23prop ;
        } Ts_P23 ;

        /* Parameterstruktur fuer Funktion "TS_DeleteLeafEntity" */
typedef struct {
        Ts_head         *ts_head ;
        short           ts24dir_id ;
        Ts_leaf_name    *ts24dist_name ;
        } Ts_P24 ;

        /* Parameterstruktur fuer Funktion "TS_AddProperty" */
typedef struct {
        Ts_head         *ts_head ;
        short           ts25dir_id ;
        Ts_leaf_name    *ts25dist_name ;
        long            ts25short_id ;
        Ts_property     *ts25prty ;
        } Ts_P25 ;

        /* Parameterstruktur fuer Funktion "TS_DeleteProperty" */
typedef struct {
        Ts_head         *ts_head ;
        short           ts26dir_id ;
        Ts_leaf_name    *ts26dist_name ;
        long            ts26short_id ;
        short           ts26prid ;
        } Ts_P26 ;

        /* Parameterstruktur fuer Funktion "TS_ChangeProperty" */
typedef struct {
        Ts_head         *ts_head ;
        short           ts27dir_id ;
        Ts_leaf_name    *ts27dist_name ;
        long            ts27short_id ;
        Ts_property     *ts27prty ;
        } Ts_P27 ;

        /* Werte der Eigenschaft "Transportadresse" */

        /* der Wert einer Transportadresse entspricht den Festlegungen */
        /* der ICMX(L)-Beschreibung (siehe "cmx.h"  -> "t_address/t_partaddr") */

        /* Werte der Eigenschaft "NEABX-Service" */
                                        /* Boolsche Variable mit */
                                        /* TS_TRUE = NEABX-Service erforderlich */
                                        /* TS_FALSE = NEABX-Service nicht erforderlich */
```

```
        /* Werte der Eigenschaft "Routinginformation" */
typedef short Ts_route ;                /* CC-Index (0 - 16) */

        /* Werte der Eigenschaft "lokaler Name - T-Selektoren" */
                                        /* siehe "cmx.h"  -> "t_myname" */

        /* Werte der Eigenschaft "Transportsystem" */
typedef char Ts_system ;
#define TS_NEA  0                       /* NEA-Transportsystem */
#define TS_ISO  1                       /* ISO-Transportsystem */
#define TS_LAN  2                       /* LAN-Transportsystem */
#define TS_STA  3                       /* Stationskopplung */

        /* Werte der Eigenschaft "Geraeteprotokoll" */
typedef char Ts_gtype ;
#define TS_NOGTYPE      0               /* kein Geraeteprotokoll */
#define TS_SS8110       1               /* Schreibstation 8110 (Typ11) */
#define TS_DSS9750      2               /* Datensichtstation 9750 (Typ57) */
#define TS_DRS8122      3               /* Druckerstation 8122 (Typ43) */

        /* Alle TNSX-Funktionen returnieren short. */
short   ts11_read_leafs () ;
short   ts12_read_children () ;
short   ts13_read_properties () ;
short   ts21_create_non_leaf_entity () ;
short   ts22_delete_non_leaf_entity () ;
short   ts23_create_leaf_entity () ;
short   ts24_delete_leaf_entity () ;
short   ts25_add_property () ;
short   ts26_delete_property () ;
short   ts27_change_property () ;
short   ts_cntrl () ;
```

```
/*****************************************************************************
 *                                                                           *
 *      SIEMENS AG Muenchen, D ST DF 211                                     *
 *                                                                           *
 *      HEADER-FILE FUER DEN NEABX-SERVICE (neabx.h)                         *
 *      I C M X ( N E A )    V 2 . 1                                         *
 *                                                                           *
 *      Datum der letzten Aenderung:  2.12.87                                *
 *                                                                           *
 *****************************************************************************
 *                                                                           *
 *      HEADER-FILE FUER DEN NEABX-SERVICE (neabx.h)                         *
 *      ===========================================                          *
 *                                                                           *
 *      fuer die Library Schnittstelle ICMX(NEA).                            *
 *                                                                           *
 *      Die Schnittstelle ICMX(NEA) realisiert den Migrations-Service        *
 *      fuer die Kommunikation mit Anwendungen die ueber ein                 *
 *      NEA-Transportsystem erreicht werden.                                 *
 *                                                                           *
 *      ICMX(NEA) setzt auf die allgemeine Library Schnittstelle ICMX(L)     *
 *      zum Transportsystem in SINIX auf.                                    *
 *                                                                           *
 *****************************************************************************/

/*
 * Returnwerte fuer NEABX-Aufrufe
 */

#define X_OK            T_OK
#define X_ERROR         T_ERROR
#define X_NOTFIRST      T_NOTFIRST
#define X_DATASTOP      T_DATASTOP
#define X_XDATSTOP      T_XDATSTOP
#define X_REPEAT        0x27
#define X_IMPOSSIBLE    0x28

/*
 * Konstanten fuer das NEABX-Protokoll
 */

#define X_ETX           0x00
#define X_ETB           0x40
#define X_ETXEOT        0x80
#define X_ETBEOT        0xc0

#define X_EBCDIC        0x00
#define X_TRANS         0x00
#define X_ASCII         0x20
#define X_UNDEF         (-1)

#define X_NOBX          0x01
#define X_NEABX         0x80

/*
 * Konstanten fuer Optionsnummern
 */

#define X_OPTA1         T_OPTA1
#define X_OPTC1         T_OPTC1
#define X_OPTC2         T_OPTC2
#define X_OPTI1         T_OPTI1
#define X_OPTD1         3

/*
 * Konstanten fuer den x_event-Aufruf
 */

#define X_NOEVENT       T_NOEVENT
#define X_DATAIN        T_DATAIN
#define X_DATAGO        T_DATAGO
#define X_XDATIN        T_XDATIN
#define X_XDATGO        T_XDATGO
#define X_CONIN         T_CONIN
#define X_CONCF         T_CONCF
#define X_DISIN         T_DISIN
#define X_REDIN         T_REDIN
#define X_REPCRQ        0x77
#define X_REPCIN        0x78
#define X_REPCCF        0x79
```

```
/*
 * NEABX-Fehlerwerte bei x_error
 */
/* Abkuerzungen:                                                        */
/* =============                                                        */
/* PE Protokollelement,                                                 */
/* Li Laengen-Indikator Protokoll-Byte                                  */
/* RK Rechnerkopplung;                    STK Stationskopplung          */

#define X_NOERROR  T_NOERROR  /* kein Fehler */
#define X_DIDSSSTK      0x01  /* Partner hat weder ETB noch ETX bei          */
                              /* DSS-Anschluss angegeben (x_datain) */
#define X_NOTNEABX      0x02  /* nicht im X_NEABX Modus */
#define X_MAXDAT        0x03  /* Beim Verbindungsaufbau mehr als */
                              /* X_MSG_SIZE Benutzerdaten erhalten */
#define X_BADLEN        0x04  /* Ungueltige Datenpufferlaenge */
#define X_BADTRANS      0x05  /* Ungueltiges Transportsystem (im TNS-Eintrag) */
#define X_BADKOPP       0x06  /* Ungueltige Kopplungsart (im TNS-Eintrag) */
#define X_BADTABLE      0x07  /* Kein Tabelleneintrag vorhanden */
#define X_BADPROT       0x08  /* Falsches Protokoll-Identifik.-Byte (bei t_conin) erhalten */
#define X_DIDSSCDE      0x09  /* Bei DSS-Anschluss Daten nicht in EBCDI-Code (x_datain) */
#define X_SENDQUIT      0x0a  /* Fehler beim Senden eines QUITTUNG-PE (x_datain) */
#define X_XSNDQUIT      0x0b  /* Fehler beim Senden eines QUITTUNG-PE */
                              /* fuer Vorrangdaten (x_datain) */
#define X_NOOPTDSS      0x0c  /* kein x_opt bei DSS-Kopplung angegeben */
#define X_BADXCODE      0x0d  /* Falscher Transmission-Code angegeben oder erhalten */
#define X_BADPRPI       0x0e  /* Unbekanntes PI-Protokoll-Identifikations-Byte erhalten */
#define X_NOTCNPE       0x0f  /* CONNECT-PE erwartet, aber nicht erhalten */
#define X_NOTCNATT      0x10  /* CONNECT-ATTENTION - PE erwartet, aber nicht erhalten */
#define X_NOTDTPE       0x11  /* DATA-PE erwartet, aber nicht erhalten */
#define X_QUITPE        0x12  /* x_datain hat ein QUITTUNGS-PE erhalten */
#define X_DATAPE        0x13  /* x_datain hat nicht erwartetes DATA-PE erhalten*/
#define X_BADPVBYTE     0x14  /* Fehlerhaftes Protokoll-Versions-Byte im NEABX Protokoll erhalten */
#define X_NONEBYTE      0x15  /* Bei DSS-Anschluss kein NEABX in der Datenphase vereinbart */
#define X_BADDTPELI     0x16  /* Falsche Laenge in Li-Byte erhalten (x_datain) */
#define X_BADSTRUKT     0x17  /* Falsche Struktur-Angabe (x_optd1) */
#define X_DS_RDLF       0x18  /* Fehler beim Aufruf der TNS-Funktion ds1_001_read_leafs() */
#define X_BADREDIR      0x19  /* Ungueltige Verbindungsumlenkung */
#define X_BADMSGLEN     0x2a  /* Ungueltige Laenge der Benutzernachricht */
#define X_NOXDRDSS      0x2b  /* DSS darf x_xdatrq nicht aufrufen */
#define X_NOXDIDSS      0x2c  /* DSS darf x_datin nicht aufrufen */
#define X_WREPBIT       0x2d  /* Falscher Eintrag im Repeat-Puffer */
#define X_NOINFO        0x2e  /* Keine TIDU-Laenge bestimmbar */
#define X_BADXREAD      0x2f  /* t_xdatin liefert Wert groesser Null */
#define X_QOVERFLOW     0x30  /* Transportquittung kann nicht mehr gespeichert werden */
#define X_NOOPT         0x31  /* Kein x_opt Pointer angegeben */
#define X_WPARAMETER    0x32  /* Falscher Par.: zB. falsche x_optnr angegeben */

#define X_NVERR1        0x33  /* unzulaessige Eingabeparam.(x_neavi, x_neavo):*/
                              /* Nullangaben in x_udatal, x_udatap, *x_udatal zu kurz, */
                              /* x_neavo: Reserve fuer NEABX-Header(8By) nicht beruecksichtigt */
                              /* x_neavo: bei Rk. OPCH-Laenge=1 nicht beruecksichtigt */
#define X_NVERR2        0x34  /* unzulaessige Optionsnummer (x_neavi, x_neavo)*/
#define X_NVERR3        0x35  /* unzulaessige Laengen in NEABV-Nachricht (x_neavi/o*/
#define X_NVERR4        0x36  /* unzulaessige Ang. zu x_prot/x_init (x_neavo) */
#define X_NVERR5        0x37  /* unzulaessige Ang. zu x_opch/x_bvmsg/x_npw (x_neavo)   */
                              /* x_neavo: Laengenangaben zu gross (Maxima: opch=8, bvmsg=80, npw=8) */

#define X_FOPENERR      0x3a  /* Fehler beim Datei-Eroeffnen: fopen neatr.<pid> */
#define X_WXOPT         0x3b  /* Falsche x_opt-Ang.: z.B. x_opt != NULL oder  */
                              /* x_opt, obwohl ohne NEABX;                    */
                              /* x_opt, obwohl 2. u. folg. Dateneinheiten, ...*/

#define XC_ERR1         0x41  /* NEABX-COBOL: X-CONIN-FKT: Falscher Par 6 ;   */
#define XC_ERR2         0x42  /* NEABX-COBOL:                                 */
#define XC_ERR3         0x43  /* NEABX-COBOL:                                 */
#define XC_ERR4         0x44  /* NEABX-COBOL:                                 */

/*
 *  NEABX-Fehlertyp, -Fehlerklasse
 * ( higher-byte des Returnwertes von x_error() ).
 */

#define X_BX1           T_BX1   /* NEABX-Fehlertyp: reserviert */
#define X_BX2           T_BX2   /* NEABX-Fehlertyp: Systemfehler */
#define X_BX3           T_BX3   /* NEABX-Fehlertyp: v. NEABX erkannter Fehler */
#define X_NEAERR        0x0b00  /* NEABX-Fehlerklasse */


/*
 * Werte fuer Parameter x_apmode
```

```
 */
#define X_ACTIVE        T_ACTIVE        /* nur aktiver Verbindungsaufbau */
#define X_PASSIVE       T_PASSIVE       /* nur passiver Verbindungsaufbau */
#define X_REDIRECT      T_REDIRECT      /* nur Verbindungsumlenkungen */

/*
 * Werte fuer Parameter x_cmode (x_event)
 */
#define X_WAIT          T_WAIT          /* Warten auf ein Ereignis */
#define X_CHECK         T_CHECK         /* Pruefen, ob ein Ereignis eingetroffen ist */

/*
 * Werte fuer Parameter x_xdata, x_timeout
 */
#define X_YES           T_YES           /* Vorrangdaten erwuenscht */
#define X_NO            T_NO            /* keine Vorrangdaten bzw. keine Ueberwachung */

/*
 * Werte fuer Parameter chain
 */
#define X_MORE          T_MORE          /* weitere Daten in der TSDU */
#define X_END           T_END           /* keine weiteren Daten in der TSDU */

/*
 * Laengendefinitionen
 */
#define X_MSG_SIZE      T_MSG_SIZE      /* Max. Nachr.Lng. b. Verbdg.Aufbau */
#define X_RED_SIZE      T_RED_SIZE      /* Max. Nachr.Lng. b. Verbdg.Umlenkung*/
#define X_EXP_SIZE      T_EXP_SIZE      /* Max. Nachr.Lng. b. Vorrangdaten */
#define X_DRQPHL        5               /* NEA-Protokoll-Hdr.Lng. bei datarq */
#define X_RED_PL        5               /* x_red-NEA-Protokoll-Laenge V1     */


/*
 * Parameter der NEABV-Funktionen: x_neavi u. x_neavo
 */

/* Optionsnummern Verbindungs-Kopplungsart (x_optnr):
 * in neavi() und neavo():
 */
#define X_OPTRK 'R'  /* Optionsang. Kopplung: Rechnerkopplung    */
#define X_OPTSK 'S'  /* Optionsang. Kopplung: Stationskopplung   */

/* Wertebereich zur NEABV-Initiative zur Datenuebertragung (x_init)
 * in neavi() und neavo():
 */
#define X_INITRQ 0   /* Initiative z. Datenuebtrgng (x_init): Partnervorschl.*/
#define X_INITOK 0   /* Einverstaendnis mit Initiative z. Datenuebtrgng  */
#define X_MYINIT 1   /* Initiative z. Datenuebertragung (x_init): eigene */


/*
 * Strukturen
 */

struct x_opta1 {                /* x_attach    */
        int x_optnr;            /* Options-Nr. */
        int x_apmode;           /* Anwendungs-Mode */
        int x_conlim;           /* max. Anzahl der Verbindungen */
};

struct x_optc1 {                /* x_conrq, x_conin, x_conrs, x_concf */
        int x_optnr;            /* Options-Nr. */
        char *x_udatap;         /* Datenpuffer */
        int x_udatal;           /* Laenge des Datenpuffers */
        int x_xdata;            /* Vorrangdaten-Auswahl */
        int x_timeout;          /* Inaktiv-Zeit */
        char x_passwd[4];       /* Passwort Verbindungsaufbau einschl. '\0' */
        int x_prot;             /* Protokollauswahl fuer Datenphase */
};

struct x_optc2 {                /* x_disrq, x_disin, x_redrq, x_redin */
        int x_optnr;            /* Options-Nr. */
        char *x_udatap;         /* Datenpuffer */
        int x_udatal;           /* Laenge des Datenpuffers */
};

struct x_optd1 {                /* Optionsstruktur fuer x_datarq und x_datain */
                                /* x_xdatin, x_xdatrq          */
        int     x_optnr;        /* Optins-Nr. */
        int     x_code;         /* Nachrichtencode */
        int     x_strukt;       /* Nachrichtenstruktur */
};
```

```
struct x_opti1 {                /* x_info */
        int x_optnr;            /* Options-Nr. */
        int x_maxl;             /* Laenge der TIDU */
};

/*
 * Adressierungsmodi
 */

#define X_TRANSDATA     T_TRANSDATA     /* Adress-Modus fuer struct x_tdaddr */
#define X_T70           T_T70           /* Adress-Modus fuer struct x_t70addr */
#define X_MNMODE        T_MNMODE        /* Adress-Modus fuer struct x_myname */
#define X_PAMODE        T_PAMODE        /* Adress-Modus fuer struct x_partaddr */

struct x_tdaddr {
        char x_admode;          /*  = X_TRANSDATA */
        char x_pname[8];        /* TRANSDATA-Rechner-/Regionsnummer */
        char x_stname[8];       /* TRANSDATA-Stationsname */
        char x_term ;           /* Strukturendekennzeichen */
};

struct x_t70addr {
        char x_admode;          /* = X_T70 */
        char x_netadr[21];      /* Netz-Wahlinformation */
        char x_tsapid[21];      /* TSAP-Identifier */
        char x_protocolid[5];   /* X.25 'protocolid' */
};

#define X_MNSIZE T_MNSIZE

struct x_myname {
        char x_mnmode;          /* = X_MNMODE */
        char x_mnres;           /* = 0 */
        short x_mnlng;          /* Laenge des ausgefuellten Teils der Struktur*/
        char x_mn[X_MNSIZE];    /* Feld fuer die T-Selektoren */
};

#define X_PASIZE T_PASIZE

struct x_partaddr {
        char x_pamode;          /* = X_PAMODE */
        char x_pares;           /* = 0 */
        short x_palng;          /* Laenge des ausgefuellten Teils der Struktur*/
        char x_pa[X_PASIZE];    /* Feld fuer die Partneradresse */
};

union x_address {
        struct x_myname xmyname;
        struct x_partaddr xpartaddr;
        struct x_tdaddr xtdaddr;
        struct x_t70addr xt70addr;
};


/*
 *  Strukturen des NEABV - Services:
 */

struct x_optrk {        /* x_neavi / x_neavo *x_opt-Struktur fuer Rk.*/
       char    x_optnr;         /* Optionsnr. 'R', 'S'                  */
       int     x_init;          /* Initiative bei Datenuebertragung  */
       int     x_opchl;         /* Laenge der OPCH in x_opchp        */
       char   *x_opchp;         /* Pointer auf OPCH-Bereich          */
       int     x_bvmsgl;        /* Lng. d. Benutzer-Verbdgsnachricht */
       char   *x_bvmsgp; };     /* Pointer auf Ben.-Verbdgsnachricht */


struct x_optsk {        /* x_neavi / x_neavo *x_opt-Struktur fuer Sk.*/
       char    x_optnr;         /* Optionsnr.: s                        */
       int     x_opchl;         /* Laenge der OPCH in x_opchp        */
       char   *x_opchp;         /* Pointer auf OPCH                  */
       int     x_bvmsgl;        /* Lng. d. Benutzer-Verbdgsnachricht */
       char   *x_bvmsgp;        /* Pointer auf Ben.-Verbdgsnachricht */
       int     x_npwl;          /* Laenge des Netzpasswortes         */
       char   *x_npwp; };       /* Pointer auf Netzpasswort          */


/*
 * Definition der NEABX-Aufrufe
 */

int     x_attach();
```

```
int     x_conrq();
int     x_conin();
int     x_conrs();
int     x_concf();
int     x_datago();
int     x_datain();
int     x_datastop();
int     x_datarq();
int     x_detach();
int     x_disrq();
int     x_disin();
int     x_error();
int     x_event();
int     x_info();
int     x_redin();
int     x_redrq();
int     x_xdatgo();
int     x_xdatin();
int     x_xdatrq();
int     x_xdatstop();

int     x_neavi();
int     x_neavo();
```

Fachwörter und Abkürzungen

Adresse	siehe *TRANSDATA-Adresse* und *TRANSPORTADRESSE*
Aktiver Partner	Der *Kommunikationpartner*, der selbst eine *Verbindung* zu einer anderen Anwendung aufbaut.
Anwendung	siehe *Kommunikationsanwendung.*
CCP	Communication Control Program. Dieses Programm ist Voraussetzung, daß im PC *CMX-Anwendungen* ablaufen können. Es gibt je nach Anschlußart verschiedene CCPs.
CCP-LAN	*CCP* für den Anschluß des PCs an lokale Netze (local area networks). Voll ausgebildetes *Transportsystem.*
CCP-STA	*CCP* für Stationsanschluß des PCs ans Netz über DÜ-Prozeduren. Kein voll ausgebildetes *Transportsystem.*
CCP-WAN	*CCP* für den Anschluß des PCs an Fernnetze, z.B. TRANSDATA. Voll ausgebildetes *Transportsystem* in verschiedenen Ausprägungen.
CMX	Communication Method SINIX.
CMX-Anwendung	Eine *Kommunikationsanwendung*, die auf einem SINIX-Rechner abläuft. Sie entsteht, wenn sich ein *Prozeß* bei CMX anmeldet.
CMX-Konstante	Größe, die rechnerspezifisch für CMX vorgegeben ist, z.B. die Länge einer *Dateneinheit.* Man kann sie mit dem Aufruf t_info abfragen.
Dateneinheit	Die Zeichenmenge, die man mit einem Aufruf t_datarq auf einmal senden oder mit einem Aufruf t_datain empfangen kann.
DCAM-Anwendung	Eine *Kommunikationsanwendung* im BS2000, die die Zugriffsmethode DCAM benutzt (Data Communication Access Method).

Eigenschaft	Attribut einer *Kommunikationsanwendung* im *TS-Directory*, wo sie mit ihrem *GLOBALEN NAMEN* erfaßt ist.
GLOBALER NAME	Name einer *Kommunikationsanwendung*, der sie im Netz eindeutig identifiziert. Unter dem GLOBALEN NAMEN steht eine *Kommunikationsanwendung* im *TS-Directory*.
Kommunikationsanwendung	Eine Kommunikationsanwendung ist ein Programm, das eine logische *Verbindung* zu einer anderen Anwendung aufbauen kann, um mit dieser Daten auszutauschen.
Kommunikationsmethode	Eine Zugriffsmethode auf Dienste der *Transportschicht* nach dem *OSI-Schichtenmodell*.
Kommunikationspartner	Eine *Anwendung*, die eine logische *Verbindung* zu einer anderen Anwendung unterhält und Daten mit ihr austauscht.
LOKALER NAME	*Eigenschaft* einer *Kommunikationsanwendung* im *TS-Directory* zum *GLOBALEN NAMEN*. Der LOKALE NAME muß bei der Anmeldung bei CMX angegeben werden.
Migrationsservice	Dienst in CMX zur Anpassung einer *CMX-Anwendung* an die Anforderungen bestehender *Kommunikationsanwendungen* in PDN und BS2000.
MIGRATIONSSERVICE	*Eigenschaft* einer *Kommunikationsanwendung* im *TS-Directory* zum *GLOBALEN NAMEN*. Sie bestimmt, ob die Anwendung *Migrationsservice* braucht.
Nachricht	Eine logisch zusammengehörige Datenmenge, die an einen *Kommunikationspartner* gesendet werden soll.
OSI	Open System Interconnection.

OSI-Schichtenmodell	Modell für die Kommunikation 'Offener Systeme'. Es ist in der Norm ISO 7498 beschrieben und enthält 7 Schichten. Eine davon ist die *Transportschicht.*
Partner	siehe *Kommunikationspartner.*
Passiver Partner	Der *Kommunikationspartner*, der eine *Verbindung* nicht selbst aufbaut, sondern von einer anderen *Anwendung* erhält.
PDN-Anwendung	Eine *Kommunikationsanwendung*, die in einem Kommunikationsrechner abläuft.
Prozeß	SINIX-Prozeß.
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.
Stationskopplung	Anschlußart für einen PC an das Netz. Die Anwendungen im PC sind als Stationen am Anschlußrechner generiert.
Station	Aus der Sicht des Transportservices netzweit adressierbare Endstelle des Datenkommunikationssystems.
Stationskopplung	Anschlußart für einen PC an das Netz. Die Anwendungen im PC sind als Stationen am Anschlußrechner generiert.
Stationsname	Ein Teil der *TRANSDATA-Adresse.* Der Stationsname korrespondiert mit dem *LOKALEN NAMEN* der *Anwendung.*
TIDU	Transport Interface Data Unit: *Dateneinheit*
TRANSDATA-Adresse	Eine Zeichenfolge, die eindeutig eine adressierbare Instanz im TRANSDATA-Netz kennzeichnet. Sie besteht aus *Prozessorname* und *Stationsname.*

TRANSPORTADRESSE	*Eigenschaft* einer *Kommunikationsanwendung* im *TS-Directory* zum *GLOBALEN NAMEN*. Die TRANSPORTADRESSE muß beim Verbindungsaufbau zu einem Kommunikationspartner angegeben werden.
Transport Name Server	Dienst in CMX zur Verwaltung transportsystemspezifischer Eigenschaften von *Kommunikationsanwendungen*.
Transportreferenz	Eine Nummer, die innerhalb einer Anwendung eine *Verbindung* eindeutig kennzeichnet.
Transportschicht	4. Schicht im *OSI-Schichtenmodell*. Sie wird beschrieben durch die Norm ISO 8072.
Transportsystem	Die unteren 4 Schichten im *OSI-Schichtenmodell*.
TS-Directory	Datenbank mit Informationen über die *Kommunikationsanwendungen*. Die Verwaltung des TS-Directory erfolgt durch den *Transport Name Server*.
TSDU	Transport Service Data Unit: *Nachricht*
UTM-Anwendung	Eine Teilhaberanwendung im BS2000, die auch mit anderen Anwendungen kommunizieren kann.
Verbindung, logische	Zuordnung zweier *Kommunikationspartner*, die es ihnen ermöglicht, Daten miteinander auszutauschen.

Literatur

[1] Betriebssystem SINIX
Buch 1
Benutzerhandbuch

Zielgruppe
Alle die das Betriebssystem SINIX kennenlernen wollen.
Inhalt
Beschreibung des Betriebssystems SINIX (Dateisystem, Shell, Systemverwaltung, Kommandos)

[2] **CES Buch 1**
Werkzeuge zur Programmierung
Grundlagen und Kommandos

Zielgruppe
Programmierer auf SINIX.
Inhalt
Nachschlagewerk über die SINIX-Kommandos des C-Entwicklungssystems.
Die Kommandos unterstützen die Installation und Verwaltung von Programmen auf einem SINIX-Rechner.
Beschrieben sind unter anderem:
COMPILER cc
Debugger adb
Bibliotheksverwaltung ar
Programmgenerierung lex, yacc
SCCS.
Einsatz
Programmentwicklung auf SINIX.

[3] **CES Buch 2**
Werkzeuge zur C-Programmierung
Systemaufrufe, C-Funktionen und Makros

Zielgruppe
C-Programmierer.
Inhalt
Nachschlagewerk über alle Systemaufrufe. Bibliotheksfunktionen und Makros, die dem C-Programmierer im C-Entwicklungssystem zur Verfügung stehen.
Einsatz
C-Programmierung

[4] **SINIX-PC-MX4**
CCP
TRANSDATA/SNA-Anschluß
Benutzerhandbuch

Zielgruppe
Kunde, der CCP selbst auf PC-MX4 installiert
Netzgenerierung des TRANSDATA-Netzes
Inhalt
Beschreibung der CCP-Standardkonfigurationen
Beschreibung der Anpassung an das Netz
CCP-Kommandos, Dienstprogramm
Generierungsprotokolle

[5] Betriebssystem SINIX
CCP
PC-X/PC-X10
Benutzerhandbuch

Zielgruppe
Kunde, der ein CCP-Produkt auf PC-X bzw. PC-X10 installiert;
Systemverwalter.
Inhalt
Installation, Konfigurierung, In- bzw. Außerbetriebnahme, Generieren im nächsten Kommunikationsrechner und Fehlermeldungen für folgende Produkte:
CCP-STA1 und CCP-STA2

[6] Betriebssystem SINIX
CCP
PC-MX2
Benutzerhandbuch

Zielgruppe
Kunde, der ein CC-Produkt auf PC-MX2 installiert; Systemverwalter.
Inhalt
Installation, Konfigurierung, In- bzw. Außerbetriebnahme, Generieren im nächsten Kommunikationsrechner und Fehlermeldungen für folgende Produkte:
CCP-STA1 und CCP-STA2,
CCP-LAN2,
CCP-WAN1 und CCP-WAN2.
Erfassungsprogramm für TNSADMIN

[7] Betriebssystem SINIX
CCP
MX500
Benutzerhandbuch

Zielgruppe
Kunde, der ein CCP-Produkt auf MX500 installiert; Systemverwalter.
Inhalt
Installation, Konfigurierung, In- bzw. Außerbetriebnahme, Generieren im nächsten Kommunikationsrechner und Fehlermeldungen für das Produkt CCP-STA1.
Erfassungsprogramm TNSADMIN.

[8] TRANSDATA
Generierung eines Datenkommunikationssystems
Benutzerhandbuch

Zielgruppe
Systemverwalter, Netzadministrator, PDN-Generierer
Inhalt
KOGS-Sprache, Netzbeschreibung, Struktur von KOGS-Programmen
Einsatz
Verarbeitungsrechner mit BS2000

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

Stichwörter

SIEMENS

Siemens MX2,MX300,MX500

CMX (SINIX)

Version 2.1A
Juni 1988

Freigabemitteilung

U30000-J566

Herausgegeben vom Bereich
Daten- und Informationstechnik
Postfach 83 09 51, D-8000 München 83

Siemens Aktiengesellschaft

I N H A L T S V E R Z E I C H N I S

1. Allgemeines

Hiermit wird CMX V2.1A (Communication Method SINIX) für das Betriebssystem SINIX ab V5.1 freigegeben.

CMX V2.1A ein Pflichtprodukt, das beim Einsatz eines der angebotenen CCP-Produkte (s.Kap. 1.5.3) bezogen werden muß.

CMX bietet, als Zugriffssystem zur Transportschicht des ISO-Referenzmodells, Anwendungsprogrammen in SINIX eine einheitliche Programmschnittstelle zu den Diensten der Transportschicht gemäß DIN ISO 8072. Je nach zur Verfügung stehenden CCP kann CMX im Modus Rechner- und Stationskopplung eingesetzt werden. Im Modus Stationskopplung sind Funktionseinschränkungen von CMX zu berücksichtigen, die im Manual zu CMX V2.1A beschrieben sind.

Teilnehmer in SINIX können mit Datenstationen und Anwendungen unter den Betriebssystemen BS2000, PDN oder SINIX in einem entfernten System kommunizieren.

Die Schnittstelle ist in Form von Bibliotheksroutinen für das att- und sie-Universum realsiert.

Mit dem Transport-Name-Service bietet CMX V2.1A einen Dienst der die CMX-Anwendung von der Konstruktion der Transportadressen entbindet. Diese werden mit C-Funktionen aus dem dem TS-Directory gelesen. Die mit den CCP's ausgelieferte Diskette TNSADM enthält ein Menüsystem, mit dem sich die Adressen im TS-Directory einfach verwalten lassen.

Die ebenfalls in der CMX-Bibliothek enthaltenen Funktionen des NEA-Migrationsservices NEABX erlauben die Programmierung von SINIX-Programmen, die NEA-spezifische, in einem Transportdienst nach ISO 8072 aber nicht enthaltene Dienstelemente unterstützen müssen. Ein solches Programm ist nur sinnvoll, wenn es mit BCAM- oder PDN-Anwendungen kommuniziert, die solche Dienstelemente voraussetzen und nicht umgestellt werden können.

1.1. Bestelldaten

Die Software müssen Sie bei der für Sie zuständigen Vertriebsgesellschaft über das Verfahren ADVOKAD bestellen.

CMX ist ein Lizenzprodukt und es gelten die allgemeinen Bestimmungen zum Kauf-, Wartungs-, Softwareüberlassungs- und Softwarebetreuungsvertrag.

1.2. Änderungen

1.2.1. Erweiterungen

1.2.2. Entfallene Funktionen

1.2.3. Inkompatibilitäten

1.3. Übergabeformate

1.3.1. Inhaltsverzeichnis des Übergabemediums

Die CMX V2.1A wird auf einer 5 1/4 Zoll-Diskette mit der Kennung CMX zur Verfügung gestellt.

Inhalt der Diskette:
Die Diskette CMX enhält eine Bibliothek von C-Funktionen, jeweils für das att- und sie-Universum, Header-Dateien für den Gebrauch von CMX, TNS und NEABX, den tnsxd-Prozeß (TNS-Zugriffs-Prozeß), sowie Traceprogramme zur Diagnose der Schnittstelle zu den unterlagerten CCP's. Vom unterlagerten CCP unabhängiges CMX-, TNS-Traceprogramm und ein TNS-Programm zur Abfrage von TNS-Einträgen auf Kommandoebene sind ebenfalls enthalten.

1.4. Dokumentation

1.4.1. Verzeichnis der Druckschriften

Zum Lieferumfang gehört das Manual:

CMX - Kommunikationsmethode in SINIX
Bestellnummer U2405-J-Z95-4

Informationen zum CCP finden Sie in:

Allgemeines:
CCP - Basis-Software zur Kommunikation ab Sinix V5.1
Grundmanual
Bestellnummer U3641-J-Z95-1

Spezielles:
CCP - Produktmanuale

1.5. Technische Daten der Freigabe

1.5.1. Minimale Anlagenausstattung

Der Einsatz von CMX erfordert keine zusätzliche Hardwarekonfiguration. Jedoch sind die Bedingungen zum Einsatz der verschiedenen CCP-Produkte zu beachten (Konfiguration der Communication-Controller CC und minimale Anlage). Vergleichen Sie dazu die Freigabemitteilung zu SINIX und zu den CCP-Produkten, die Sie einsetzen wollen.

1.5.2. Speicherbedarf der einzelnen SW-Lieferbestandteile

s. Inhaltsverzeichnis der Diskette.

1.5.3. Schnittstellenabhängigkeiten

Die folgende Softwarekonfiguration ist Voraussetzung für die Installation und Gebrauch von CMX V2.1A:
SINIX ab V5.1
und mindestens 1 Produkt aus:
CCP-LAN2 ab V1.0
CCP-STA1 ab V2.1
CCP-STA2 ab V1.1
CCP-WAN1 ab V1.0
CCP-WAN2 ab V1.0
CCP-WAN3 ab V1.0
CCP-WAN4 ab V1.0
CCP-WAN5 ab V1.0
CCP-WAN6 ab V1.0

Kommunikation mit Partneranwendungen im BS2000 oder PDN, in Abhängigkeit von CCP-Einsatz und erforderlicher PDN-Version, siehe Freigabemitteilungen der CCP's.

1.6. Allgemeine Hinweise

1.6.1. Hinweise zur Produktinstallation

Spielen Sie die CMX-Diskette über das admin-Menü "Installation von Softwareprodukten" im sie-Universum ein. Vorhandene Dateien mit gleichen Namen, wie auf der Diskette verzeichnet, werden überschrieben.

1.6.2. Hinweise zum Einsatz des Produktes

CMX, bzw. die Programmschnittstelle zu den Diensten der Transportschicht gemäß DIN ISO 8072, steht im sie- und att-Universum zur Verfügung.

CMX-Anwendungen können Verbindungen zu BS2000-Anwendungen (DCAM,TIAM u. UTM),PDN-Anwendungen (APS), sowie anderen CMX-Anwendungen unterhalten. Die Programmierung ist unabhängig vom unterlagerten CCP, da die Abbildung zwischen Anwendungsnamen und den CCP-spezifischen Transportadressen von den Diensten des Transport-Name-Service TNS übernommen wird.

Mit CMX V2.1A ist ein Parallelbetrieb von Stationskopplung mit CCP-STA1 und Rechnerkopplung möglich, während über CCP-STA2 innerhalb eines Prozesses nur 1 Anwendung mit 1 Verbindung betrieben werden kann.

1.7. Verhalten im Fehlerfall

Wenn Sie sicher sind, daß kein Anwenderfehler vorliegt, sollten Sie das CMX-Programm sicherstellen, mit dem sich der Fehler reproduzieren läßt.

Bitte erstellen Sie einen CMX-Trace, sowie einen Trace des unterlagerten CCP's und übergeben Sie diese Ihrem zuständigen System- bzw. Wartungsdienst. Ggf. ist auch ein Trace des tnsxd-Prozesses (TNS-Dämon) beizulegen. Die Bedienung der Traces ersehen Sie aus dem CMX-Manual.

2. Hardware

siehe Freigabemitteilungen der gewünschten CCP-Produkte.

3. Software

3.1. Software-Lieferbestandteil libcmx.a

3.1.1. Funktionen

CMX besteht aus einer Bibliothek von C-Funktionen, die ein Programmierer in seinen C-Programmen verwenden kann.
Mit diesen Funktionen lassen sich
- Anwendungen an- und abmelden (t_attach, t_detach),
- Verbindungen aufbauen (t_conrq, t_conin, t_conrs, t_concf),
- Verbindungen abbauen (t_disrq, t_disin),
- Verbindungen für Normaldaten sperren und freigeben (t_datastop, t_datago),
- Verbindungen für Vorrangdaten sperren und freigeben (t_xdatstop, t_xdatgo),
- Verbindungen umlenken (t_redrq, t_redin),
- Normaldaten senden und empfangen (t_datarq, t_datain),
- Vorrangdaten senden und empfangen (t_xdatrq, t_xdatin),
- Ereignisse entgegennehmen (t_event),
- die maximale Länge einer Dateneinheit abfragen (t_info),
- Fehlercodes vom System abholen (t_error).

Die NEABX-Funktionen unterscheiden sich von CMX-Funktionen durch Angabe anderer Parameter und beginnen in der Syntax mit 'x_...' anstatt von 't_...'.

Über den Unterschied zwischen Programmen, Prozessen und Anwendungen informiert im CMX-Manual Kapitel 1-3.

Grenzwerte:

	MX500 Rechnerk. + CCP-STA1	MX300 Rechnerk.	MX300 CCP-STA2 + CCP-STA1	MX2 Rechnerk.	MX2 CCP-STA2 + CCP-STA1
1.) pro Rechner:					
Prozesse, die Anwendungen bei CMX anmelden koennen	128	96	32	64	32
Anzahl der Anwend.	128	96	16	64	16
Anzahl der Anmeld.	128	96	32	64	32
Anzahl der Verbind.	128	96	16	64	16
2.) pro Prozeß:					
Anzahl der Anmeld.	128	96	1	64	1
Anzahl der Verbind. pro Anmeldung	Rechnerkoppl.: 128 CCP-STA1/STA2: 1	Rechnerkoppl.: 128 CCP-STA1/STA2: 1		Rechnerkoppl.: 128 CCP-STA1/STA2: 1	

3.) pro CC (Communication Controller):
siehe Freigabemitteilung der CCP's

Anmerkung:
Bei Ermittlung der Grenzwerte für eine bestimmte Konfiguration sind die jeweils relevanten Grenzwerte aller Gruppen maschinen-, prozeß-spezifisch, sowie die CCP-spezifischen Grenzwerte einzubeziehen. Die dabei auftretenden niedrigsten Grenzwerte sind ausschlaggebend.

- Allgemeines:

Bei der Funktion t_attach() ist zu beachten, daß Sie den zu übergebenden LOKALEN NAMEN der lokalen Anwendung mit der TNS-Funktion ts13_read_properties ermitteln müssen.

Vor dem t_conrq()-Aufruf muß der an die Funktion zu übergebende LOKALE NAME für die lokale Anwendung und die Transportadresse für die entfernte Anwendung wiederum mit Hilfe der TNS-Funktion ts13_read_properties gefüllt werden.

Eine positive Antwort des Partners auf einen Verbindungsaufbauwunsch nimmt die Anwendung mit t_concf() entgegen. Erst jetzt ist die zugehörige Transportreferenz etabliert und es kann z.B. t_info() aufgerufen werden.

Eine Anwendung, die ein T_CONIN erhalten hat, kann sich über ts11_read_leafs() über den GLOBALEN NAMEN der rufenden Anwendung informieren, bevor sie die Verbindung annimmt.

- Einsatzfall Stationskopplung:

Es lassen sich grundsätzlich nur t_attach()-Aufrufe erfolgreich absetzen, wenn der dem GLOBALEN NAMEN zugeordnete LOKALE NAME einem CMX-Stationsnamen entspricht. Die Wahl des CMX-Stationsnamen ist frei. Man kann z.B. auch die entsprechenden Stationsnamen aus der PDN-Generierung des Anschlußrechners übernehmen. Wird eine Verbindung zu einem fernen Partner aufgebaut, der über Stationskopplung in das TRANSDATA-Netz eingebunden ist, muß als Stationsname innerhalb der TRANSDATA-Transportadresse der Stationsname aus der PDN-Generierung angegeben werden. Beispiele hierzu finden Sie in Kapitel 6 des Produktmanuals zu CCP-STA1 bzw. CCP-STA2.

Auftretende Leitungsfehler führen bei einer bestehenden Verbindung bei:
CMX-STA1: Anzeige von T_DISIN mit reason 483
CMX-STA2: Anzeige von T_ERROR beim t_event()-Aufruf
Besteht die Verbindung noch nicht, so äußern sich auftretende Leitungsfehler bei:
CMX-STA1: werden nicht gemeldet
CMX-STA2: Anzeige von T_NOEVENT beim t_event()-Aufruf.

Eine fehlerhafte Abstimmung zwischen der PDN-Generierung im Anschlußrechner und der Konfigurierung von CCP-STA führt möglicherweise bei:
CMX-STA1: Anzeige von T_DISIN mit reason 485 im Anschluß an einen erfolgreichen t_conrq()-Aufruf
CMX-STA2: Fehler 108 beim t_conrq()-Aufruf.

Beim nicht regelrechten Verbindungsabbau oder Ausfall des Rechners ist zu beachten, daß der Anschlußrechner nichts von diesen Ausfällen merkt; es kommt keine Verbindungsabbaumeldung beim Partner an. Falls dieser weitersendet, bauen die PDN-Module YMPCX und YMPCX1 die Verbindung ab.

Bei Stationskopplung sind die im Produkt CMX V2.1A enthaltenen Funktionen t_datastop() und t_datago() mit äußerster Sorgfalt anzuwenden. Werden von der Partneranwendung Daten zugestellt und nicht abgeholt, da t_datastop() vorliegt, so kann der

Datenverkehr auf den anderen Datenkanälen des CCP's behindert werden. Weiterhin gehen die Daten verloren, wenn sie nicht vor Ablauf einer gewissen Zeit abgeholt werden.

Als Grundregel sollte in CMX-Programmen unter Stationskopplung so oft als möglich t_event() aufgerufen werden, damit solche Situationen vermieden werden.

3.1.2. Literaturhinweise

- Hinweis zum Manual zu CMX V2.1, Kapitel 8.3:

1.) Einschränkungen zu t_datarq():
Die maximale Länge einer Dateneinheit (TIDU) ist dynamisch durch t_info() erfragbar (z.Zt. 440 Bytes). Es dürfen max. 10 TIDU's miteinander verkettet werden, d.h. die Abfolge

t_datarq(...,T_MORE) } 9 *
t_datarq(...,T_MORE) }
t_datarq(...,T_END)

ist erlaubt, sofern die geltenden Einschränkungen bzgl.der Gesamtlänge der Nachricht befolgt werden.

2.) Einschränkungen zu t_datain(),t_datastop():
Empfangsdaten, die von der entsprechenden Kommunikationsanwendungen nach einer bestimmten Zeit nicht abgeholt werden, werden verworfen. Diese Zeitspanne läßt sich nicht, wie im CMX-Manual dargestellt, durch Einstellung des WABT-Zählers beeinflussen, sondern ist fest vorgegeben. Einzelheiten finden Sie in der Freigabemitteilung zu CCP-STA1 V2.1. Die fehlerhafte Darstellung wurde erst nach Drucklegung des CMX-Manuals bemerkt. Wir bitten um Ihr Verständnis.

3.1.3. Anwendung neuer Funktionen

3.1.4. Anwendung bisheriger Funktionen

3.1.5. Hinweise

Das Signal SIGTERM sollte in CMX-Programmen nicht verwendet werden. Die CMX-Funktionen verwenden es intern.

Beim 1. t_attach() pro Prozeß wird ein Filedeskriptor belegt, so daß dem Benutzer 1 Filedeskriptor weniger als möglich zur Verfügung steht (s. Manual). Bei Zugriffen auf den Transport-Name-Service werden temporär 3 Filedeskriptoren belegt.

Der durch CMX realisierte Transportservice ist nicht deckungsgleich mit dem NEA-Transportservice,z.B. bietet ISO 8072 nicht:
- Passwort beim Verbindungsaufbau
- Benutzerdaten beim Verbindungsaufbau > 32 Byte
- Nachrichtenstrukturierung (ETX/ETB)
- Mitteilung über den verwendeten Nachrichtencode
- Sequenznummer beim Nachrichtenaustausch
- Beantwortung angeforderter Transportquittungen.

Die Kommunikation von CMX-Anwendungen mit Anwendungen (DCAM, APS,UTM,TIAM), die von CMX nicht unterstützte Dienstelemente anfordern, ist nur mit einem CMX-Programm möglich, das die Funktionen des NEABX-Migrationsservice benutzt. Insbesondere ist die Datenübertragung durch CMX codetransparent.

3.2. Software-Lieferbestandteile cmx.h, tnsx.h und neabx.h

3.2.1. Funktionen

Will man in einem C-Programm CMX-Funktionen oder -Strukturen benutzen, muß man die Dateien cmx.h und tnsx.h per Include-Anweisung einschliessen. Sie enthalten u.a. Erklärungen zu den Returncodes der CMX- und TNS-Funktionen. Ist der Returncode einer CMX-Funktion nicht in cmx.h enthalten, so ist die Fehlerbedeutung in errno.h zu suchen. Fehlerwerte aus TNS-Funktionen sind in tnsx.h enthalten.
Werden in einem CMX-Programm NEABX-Funktionen aufgerufen, so ist zusätzlich die Datei neabx.h per include-Anweisung einzuschliessen.

3.2.2. Literaturhinweise

Zur Datei /usr/include/errno.h finden Sie Näheres im CES-Buch 2 bei der Funktion perror() auf Seite 2-280ff..

3.2.3. Anwendung neuer Funktionen

3.2.4. Anwendung der bisherigen Funktionen

3.3. Software-Lieferbestandteil tnsxd

3.3.1. Funktionen

Das Programm tnsxd verwaltet als Dämon den Zugriff auf die Daten, die im TS-Directory enthalten sind. Er wird beim Systemstart automatisch gestartet.

3.3.2. Literaturhinweise

3.3.3. Anwendung neuer Funktionen

3.3.4. Anwendung der bisherigen Funktionen

3.3.5. Hinweise

3.4. Traceprogramme dstrace, cmxt, cxlt, cxwt

s. unter 1.7. Verhalten im Fehlerfall und Manual.