CMX V6.0

Anwendungen programmieren

Ausgabe Juni 2003

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Für Ihre Kommentare stehen Ihnen Fax-Formulare auf den letzten Seiten dieses Handbuchs zur Verfügung.

Dort finden Sie auch die Adressen der zuständigen Redaktion.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2000

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2000 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright und Handelsmarken

Dieses Handbuch wurde
auf chlorfrei gebleichtem
Papier gedruckt.

weitere Kapitel ▶

Inhalt

1 Einleitung

1.1 Kurzbeschreibung des Produkts CMX

Das Transportzugriffssystem CMX (Communications Manager UNIX) ist das Basisprodukt der Kommunikationssoftware. CMX ermöglicht die Kommunikation zwischen Anwendungen in verschiedenen Rechnersystemen. Zusammen mit den Kommunikationssteuerprogrammen CCP (Communication Control Program) übernimmt CMX die Kommunikationsaufgaben. Mit der von CMX angebotenen Programmschnittstelle lassen sich Anwendungsprogramme erstellen, die unabhängig vom Transportsystem mit anderen Anwendungen kommunizieren können.

1.2 Zielgruppen des Handbuchs

Das Manual richtet sich an den Programmierer von TS-Anwendungen zur Kommunikation (TS-Anwendung = Transport-Service-Anwendung), die aus in C implementierten Anwendungsprogrammen bestehen.

Um mit CMX arbeiten zu können, benötigen Sie Kenntnisse des Betriebssystems. Es wird die Beherrschung der Programmiersprache C und des C-Entwicklungssystems erwartet. Ferner ist für das Verständnis das Wissen über die Prinzipien und Methoden der Datenfernverarbeitung hilfreich, insbesondere über das OSI-Referenzmodell, wie in DIN ISO 7498 normiert.

1.3 Konzept des Handbuchs

Die Beschreibung des gesamten Produkts CMX umfasst zwei Benutzerhandbücher:

– CMX „Betrieb und Administration“ für Systemverwalter und Benutzer,

– CMX „Anwendungen programmieren“ für den Programmierer von TS-Anwendungen.

Das vorliegende Handbuch (CMX „Anwendungen programmieren“) beschreibt die Programmschnittstellen von CMX, d. h. alle Werkzeuge, die Sie benötigen, um selbst TS-Anwendungen zu entwickeln.

Hilfsmittel zur Diagnose, das sind Bibliotheksverfolger und Decodierprogramm für CMX-Meldungen, finden Sie in CMX „Betrieb und Administration“.

Aufbau dieses Handbuchs

Das vorliegende Handbuch ist in zwei Teile strukturiert:

Der 1. Teil ist zum Kennenlernen von CMX gedacht und soll dem Einsteiger helfen, TS-Anwendungen zu erstellen. Er beschreibt die Abbildung einer TS-Anwendung auf das Prozesskonzept Ihres Systems und die Zuordnung der Transportverbindungen zu den Prozessen der TS-Anwendung. Er zeigt die Strukturierung einer TS-Anwendung in drei Kommunikationsphasen und erläutert, wie die Funktionen der Programmschnittstellen innerhalb dieser Phasen angewendet werden. Sie erfahren, wie Sie Namen und Adressen mit Hilfe des Transport Name Service (TNSX) aus dem Adress-Verzeichnis abfragen und an CMX übergeben können, und wie Sie im Fehlerfall Diagnoseinformationen von CMX abfragen können. Zu den einzelnen Programmierschritten finden Sie Programmfragmente als Beispiele.

Der 2. Teil umfasst die Kapitel 8 und 9. Jedes dieser Kapitel beschreibt eine der CMX-Programmschnittstellen. Hier wird jeder einzelne Funktionsaufruf der jeweiligen Programmschnittstelle und seine Parameter im Detail beschrieben. Die Beschreibung erfolgt in alphabetischer Reihenfolge. Am Anfang jeden Kapitels finden Sie eine Zusammenfassung aller Informationen, die Sie zur Anwendung der Funktionen benötigen.

Bei der Beschreibung der Programmschnittstellen werden alle Möglichkeiten der Anbindung eines Rechners ans Netz (LAN und WAN) berücksichtigt. Welche Möglichkeiten Ihnen an Ihrem Rechner zur Verfügung stehen, ist abhängig von den CCs (Communication Controller) und CCPs (Communication Control Program), die an Ihrem Rechner installiert sind.

Die Programmschnittstellen sind unabhängig vom jeweiligen Betriebssystem beschrieben. Betriebssystemspezifische Besonderheiten finden Sie in der Freigabemitteilung zu CMX.

Verweise auf andere Manuale

Im Text wird mit „siehe [n]“ auf andere Manuale verwiesen, die weiterführende Informationen enthalten. Dabei ist n eine Ziffer. Unter [n] finden Sie die Titel der entsprechenden Manuale im Literaturverzeichnis zusammen mit einer kurzen Inhaltsangabe.

1.4 Programmbeispiele

Zum Produktumfang von CMX gehören Beispielprogramme zu TS-Anwendungen an ICMX(L) und ICMX(NEA). Die C-Quellcodes zu diesen Programmen sind im Dateiverzeichnis *opt/lib/cmx/demo* (für Reliant UNIX) bzw. */opt/SMAW/SMAWcmx/lib/cmx/demo* (für Solaris) abgelegt. Zusätzlich finden Sie dort ein Beispielskript, um Einträge ins TS-Directory aufzunehmen, sowie Beispielskripts zum Aufrufen der Programme.

Was diese Programme leisten, wie Sie sie übersetzen müssen und was Sie beim Aufruf der Programme beachten müssen, ist in Kommentaren am Anfang der C-Quellprogramme enthalten.

1.5 Readme-Dateien

Funktionelle Änderungen und Nachträge der aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. produktspezifischen Readme-Dateien. Sie finden sie in den jeweiligen Verzeichnissen der Pakete unter *opt/readme* (Reliant UNIX) bzw. in den jeweiligen Verzeichnsissen der Produkte unter */opt/SMAW/documents* (Solaris), sofern Ihr Systemverwalter sie installiert hat. Die Readme-Dateien können Sie mit einem Editor ansehen oder auf einem Standarddrucker ausdrucken.

2 Das Transportzugriffssystem CMX

2.1 Kommunikation zwischen TS-Anwendungen

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

Eine TS-Anwendung sollte über verschiedene Transportsysteme Verbindungen aufbauen und Daten austauschen können. Eine TS-Anwendung sollte also möglichst unabhängig von dem zugrundeliegenden Transportsystem sein. Die Transportsysteme unterscheiden sich z. B. in der Größe der Dateneinheit, die das jeweilige Transportsystem übernehmen kann, im Format der zu übergebenden Transportadresse der Partneranwendung und im Format der Adresse der TS-Anwendung im lokalen System.

CMX bietet den TS-Anwendungen aus diesem Grund eine einheitliche Schnittstelle zu den verschiedenen Transportdiensten an, die Programmschnittstelle ICMX(L). An dieser Schnittstelle stehen den TS-Anwendungen die Dienste der Transportsysteme, die den Regeln des OSI-Referenzmodells für offene Systeme entsprechen, zur Verfügung. CMX ist also ein Transportzugriffssystem.

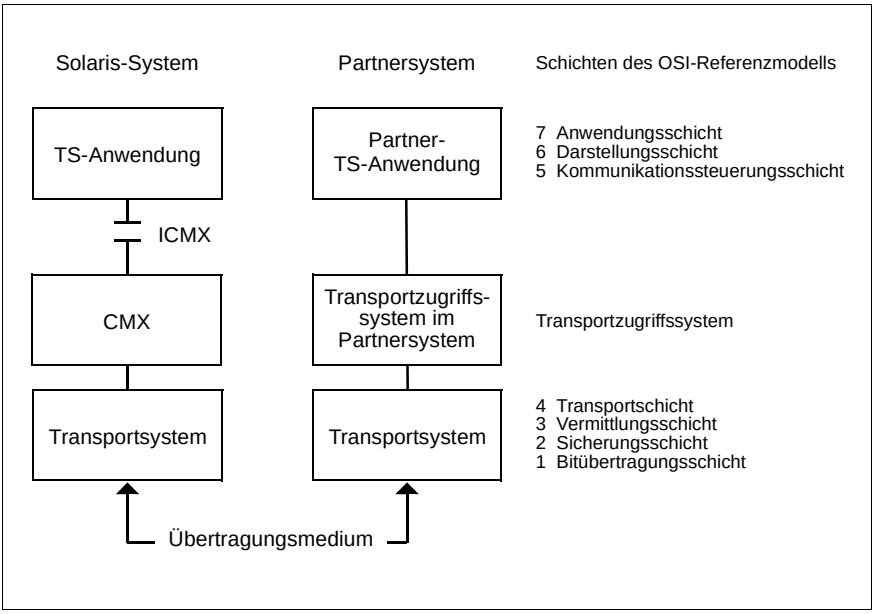

Bild 1: Das Transportzugriffssystem CMX

Eine TS-Anwendung, die die Funktionen von CMX nutzt, kann somit in einheitlicher Weise mit folgenden Partnern kommunizieren:

– mit anderen TS-Anwendungen auf demselben Rechner (lokale Kommunikation),

– mit TS-Anwendungen auf anderen Rechnern, die die Funktionen des Transportzugriffssystems CMX nutzen,

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

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

– mit TS-Anwendungen in Fremdsystemen, sofern sie die im OSI-Modell aufgeschriebenen Normen erfüllen oder sofern sie über TCP/IP mit dem Konvergenzprotokoll RFC1006 angebunden sind.

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

– seine TS-Anwendung bei CMX anzumelden,

– Transportverbindungen zu Partneranwendungen aufzubauen,

– Daten zu senden und zu empfangen,

– den Datenfluss zu regeln,

– Transportverbindungen abzubauen,

– seine TS-Anwendungen bei CMX abzumelden.

Soll eine TS-Anwendung mit einer Partneranwendung in Endsystemen mit Betriebssystem BS2000/OSD bzw. PDN kommunizieren, die noch nicht den OSI-Regeln angepasst ist und somit die umfassendere Funktionalität der NEA-Transportdienste voraussetzt, so muss sie den Migrationsservice NEABX verwenden. NEABX setzt auf den Funktionen von ICMX(L) auf und ist realisiert durch die Programmschnittstelle ICMX(NEA) von CMX.

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

2.2 Die Programmschnittstellen von CMX - eine Übersicht

CMX bietet dem Programmierer von TS-Anwendungen folgende zwei Funktionskomplexe an:

- die Funktionen zur verbindungsorientierten Kommunikation.

  Diese Funktionen umfassen lokale Dienste, die Verbindungsbehandlung und den Datenaustausch. Sie stehen über die Programmschnittstelle ICMX(L) zur Verfügung.

- den NEABX-Migrationsservice.

  Die Funktionen von NEABX erlauben es CMX-Anwendungen, mit TS-Anwendungen in Kommunikationsrechnern und Verarbeitungsrechnern der TRANSDATA-Familie, die an die OSI-Konventionen nicht angepasst sind, zu kommunizieren. Der NEABX-Migrationsservice steht über die Programmschnittstelle ICMX(NEA) zur Verfügung.

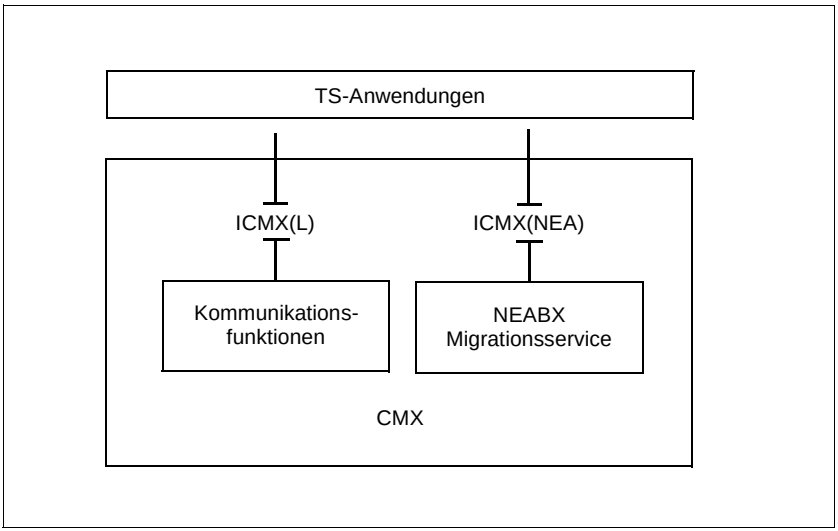

Bild 2: CMX-Progammschnittstellen

Die Programmschnittstellen von CMX sind Bibliotheksschnittstellen. Das heißt, die Funktionen von CMX stehen in Form von Moduln in einer Bibliothek zur Verfügung. Diese Moduln werden zu den Programmen der TS-Anwendung hinzugebunden. Alle Module sind in der Bibliothek */usr/lib/libcmx.so* (ein "shared object") zusammengefasst.

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

Die Programmschnittstelle ICMX(L) umfasst alle Funktionen, die eine TS-Anwendung zur Kommunikation benötigt.

Die folgenden Funktionengruppen stehen an ICMX(L) zur Verfügung:

– Funktionen zum An- und Abmelden bei CMX
– Funktionen zum Aufbau einer Verbindung
– Funktionen zum Abbau einer Verbindung
– Funktionen zum Umlenken einer Verbindung
– Funktionen zum Austausch von Daten
– Funktionen zur Fluss-Regelung
– Funktionen zur Abfrage von Informationen

Funktionen zum An- und Abmelden bei CMX

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

Funktionen zum Aufbau einer Verbindung

Dazu gehören die Funktionen zum:

– aktiven Verbindungsaufbau:

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

– passive Verbindungsannahme:

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

Funktionen zum Abbau einer Verbindung

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

Funktionen zum Umlenken einer Verbindung

Innerhalb einer TS-Anwendung kann eine Verbindung an einen anderen Prozess derselben TS-Anwendung weitergegeben (umgelenkt) werden. Die beiden Funktionen dieser Gruppe dienen dazu eine Verbindung umzulenken (redirect request) und eine Verbindung von einem anderen Prozess entgegen zu nehmen (redirect indication).

Funktionen zum Austausch von Daten

Dazu gehören Funktionen mit denen Sie:

– Normaldaten senden (data request) und empfangen (data indication) können.

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

Funktionen zur Fluss-Regelung

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

Funktionen zur Abfrage von Informationen

Diese Gruppe umfasst folgende Funktionen:

– Ein Ereignis (event) abwarten oder abholen. Ein Ereignis ist z. B. der Abbau einer Verbindung durch den Kommunikationspartner.

– Fehler (error) abfragen.

– Information (information) über CMX-Parameter abfragen.

– LOKALE und GLOBALE NAMEN, TRANSPORTADRESSEN abfragen (get local name, get name, get address).

In Kapitel 4 bis 7 wird die Verwendung der Funktionen in den Programmen einer TS-Anwendung erläutert.

2.2.2 CMX-Funktionen zur Migration (ICMX(NEA))

Die CMX-Funktionen zur Migration sind im NEABX-Migrationsservice zusammengefasst. Die Funktionen von NEABX stehen an der Schnittstelle ICMX(NEA) zur Verfügung.

Der NEABX-Migrationsservice unterstützt die Kommunikation von CMX-Anwendungen mit TS-Anwendungen in Kommunikationsrechnern mit PDN und Verarbeitungsrechnern mit BS2000/OSD, 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. 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

Dieser erweiterte Funktionsumfang wird durch das NEABX-Protokoll geboten. Aufbau und Auswertung dieses Protokolls erfolgen durch die NEABX-Funktionen. Die Funktionsaufrufe erfolgen in analoger Weise zu den in Abschnitt „CMX-Funktionen zur Kommunikation (ICMX(L))“ auf Seite 9 beschriebenen CMX-Funktionen zur Kommunikation.

Der NEABX-Migrationsservice ermöglicht einer bestehenden TS-Anwendung, die nicht den OSI-Konventionen entspricht, die Kommunikation mit einer CMX-Anwendung. Eine solche TS-Anwendung kann daher ohne Änderung ihrer Kommunikationsschnittstelle mit einer CMX-Anwendung kommunizieren. Umgekehrt ermöglicht der NEABX-Migrationsservice einer CMX-Anwendung, die erweiterte Funktionalität einer solchen TS-Anwendung zu bedienen.

Der Migrationsservice NEABX ist so lange notwendig, bis die Kommunikationsschnittstellen derartiger TS-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 aber nicht verfügbar sind.

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

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 an ICMX(L). Die ICMX(NEA)-Funktionen rufen intern ICMX(L)-Funktionen auf. Sie lassen sich daher analog in die in Abschnitt „CMX-Funktionen zur Kommunikation (ICMX(L))“ auf Seite 9 beschriebenen Funktionengruppen einteilen. In Kapitel 4 bis 7 wird die Verwendung der Funktionen in den Programmen einer TS-Anwendung erläutert. Die Funktionsaufrufe sind im Abschnitt „ICMX(NEA) - Funktionsaufrufe“ auf Seite 246 im Detail beschrieben.

2.2.3 Optionen des Systems und des Benutzers

Der Funktionsumfang der Programmschnittstellen 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 Anbindung ans Netz, 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**	**System-Option**	**Benutzer-Option**
Benutzerdaten beim Verbindungsaufbau	nein	ja	ja	ja
Benutzerdaten beim Verbindungsabbau	nein	ja	ja	ja
Vorrangdaten	ja	ja	ja	ja
Überwachung der Inaktivzeit	nein	ja	ja	ja
Verbindungslimit, Aktiv/Passiv-Modus	nein	ja	nein	ja
Benutzerreferenz der Anmeldung	nein	ja	nein	ja
Benutzerreferenz der Verbindung	nein	ja	nein	ja
Zeitschranke bei synchroner Ereignisverarbeitung	nein	ja	nein	ja
Wartezeit bei der Verbindungsumlenkung	nein	ja	nein	ja

Tabelle 1: System-Optionen von CMX

Die Systemoptionen richten sich nach dem Funktionsumfang des Transportsystems. Wenn man Optionen verwendet, die das Transportsystem oder die Kommunikationsschnittstelle der Partneranwendung nicht bietet, kommt der Verbindungsaufbau nicht zustande, oder man erhält eine Verbindungsabbauanzeige von CMX. 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 der TS-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 den Handbüchern zu den einzelnen CCP-Produkten beschrieben.

3 TS-Anwendungen

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

In den Abschnitten dieses Kapitels werden die folgenden Punkte betrachtet:

– Name und Eigenschaften einer TS-Anwendung

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

– Struktur einer TS-Anwendung

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

– Zusammenhang TS-Anwendung, Prozesse, Verbindung

  Es wird beschrieben, wie sich eine TS-Anwendung auf das Prozess-Konzept des Systems abbilden lässt, und die Zuordnung Prozess - Verbindung veranschaulicht.

– Threads und Multithreading

  Der Abschnitt gibt einen Überblick über Threads und Multithreading. Außerdem werden in diesem Kontext Verbindungen und Prozesse beschrieben, die dafür erforderlichen CMX-Bibliotheksfunktionen aufgelistet und das Übersetzen und Binden beschrieben.

3.1 Namen und Adressen von TS-Anwendungen

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

Die GLOBALEN NAMEN aller TS-Anwendungen im lokalen System und aller TS-Anwendungen in fernen Systemen, mit denen die lokalen TS-Anwendungen kommunizieren wollen, stehen in einem Adress- und Namensverzeichnis, dem TS-Directory. Im TS-Directory sind zum GLOBALEN NAMEN die Eigenschaften der TS-Anwendung gespeichert. Eigenschaften sind die Informationen über die Kommunikationspartner, die das jeweilige Transportsystem zum Aufbau einer Verbindung benötigt. Die Transportadresse einer TS-Anwendung ist z. B. eine ihrer Eigenschaften.

Innerhalb einer TS-Anwendung wird nur mit den GLOBALEN NAMEN der beiden Kommunikationspartner gearbeitet. Die TS-Anwendungen sind somit unabhängig von der transportsystemspezifischen Adressierung und von Änderungen innerhalb des Netzes. Es müssen nur die entsprechenden Eigenschaften ins TS-Directory aufgenommen oder geändert werden. Die TS-Anwendung liest mit Hilfe bestimmter Funktionsaufrufe von ICMX(L) die Eigenschaften aus dem TS-Directory und gibt diese direkt (d. h. unbesehen) an CMX weiter.

Die Verwaltung der Eigenschaften und die Vergabe GLOBALER NAMEN muss durch die TNSX-Administration erfolgen. Sie muss sicherstellen, dass die GLOBALEN NAMEN aller TS-Anwendungen voneinander verschieden sind. D. h. verschiedene TS-Anwendungen müssen unterschiedliche GLOBALE NAMEN haben.

Einen Überblick über TNS finden Sie in den Handbüchern „CMX, Betrieb und Administration“ [1] und [2].

3.1.1 Der GLOBALE NAME einer TS-Anwendung

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

Ein Anwendungsprogramm, das in mehreren Rechnern abläuft, muss mit jeweils verschiedenen GLOBALEN NAMEN adressiert werden. Der Programmname und der GLOBALE NAME der TS-Anwendung dürfen nicht verwechselt werden. Eine TS-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.

3.1.2 Eigenschaften einer TS-Anwendung

Als Eigenschaften einer TS-Anwendung werden alle Informationen über die TS-Anwendung bezeichnet, die das Transportsystem zum Aufbau der Verbindung und Verwaltung des Datentransfers benötigt. Die Eigenschaften werden im TS-Directory den GLOBALEN NAMEN zugeordnet. Welche Eigenschaften einer TS-Anwendung im lokalen System bzw. den Kommunikationspartnern (den entfernten TS-Anwendungen) zugeordnet werden können, finden Sie in den folgenden Tabellen. Beim Anmelden der lokalen TS-Anwendung bei CMX bzw. beim Verbindungsaufbau müssen Sie einige dieser Eigenschaften vom TNSX abfragen und an CMX übergeben.

Eigenschaften einer lokalen TS-Anwendung

Eigenschaft	Bedeutung der Eigenschaft
LOKALER NAME	Diese Eigenschaft benötigen Sie, um die TS-Anwendung im lokalen Endsystem bei CMX anzumelden. Der LOKALE NAME besteht aus den Adressen der TS-Anwendung im lokalen System für die verschiedenen Transportsysteme. Er ist ein Hexadezimalstring mit nichtabdruckbaren Zeichen.
USER1 USER2 USER3	Benutzer-Eigenschaften. Sie können jeder TS-Anwendung bis zu 3 dieser benutzerspezifischen Eigenschaften in freiem Format zuordnen. In ihnen können Sie für Ihre Anwendung relevante Informationen hinterlegen. CMX und TNS benutzen diese Eigenschaften nicht.

Tabelle 2: Eigenschaften einer lokalen Anwendung

Eigenschaften einer entfernten TS-Anwendung

Eigenschaft	Bedeutung der Eigenschaft
TRANSPORTADRESSE	Diese Eigenschaft besitzt als Wert die von CMX erwartete Transportadresse des Kommunikationspartners beim Verbindungsaufbau. Sie ist ein Hexadezimalstring mit nichtabdruckbaren Zeichen.
TRANSPORTSYSTEM	Diese Eigenschaft enthält als Wert den Typ des Transportsystems für die Kommunikation zu einem entfernten Kommunikationspartner. Wenn Sie eine TS-Anwendung schreiben, brauchen Sie sich mit dieser Eigenschaft nicht zu beschäftigen. CMX verwendet sie intern.
USER1 USER2 USER3	Benutzer-Eigenschaften. Sie können jeder TS-Anwendung bis zu 3 dieser benutzerspezifischen Eigenschaften in freiem Format zuordnen. In ihnen können Sie für Ihre Anwendung relevante Informationen hinterlegen. CMX und TNSX benutzen diese Eigenschaften nicht.

Tabelle 3: Eigenschaften einer fernen Anwendung

Im Folgenden werden die Eigenschaften TRANSPORTADRESSE und LOKALER NAME und ihre Bedeutung näher beschrieben.

3.1.3 Eigenschaften LOKALER NAME und TRANSPORTADRESSE

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

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

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

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

Das folgende Bild veranschaulicht den Zusammenhang zwischen LOKALEM NAMEN, TSAP und TRANSPORTADRESSE.

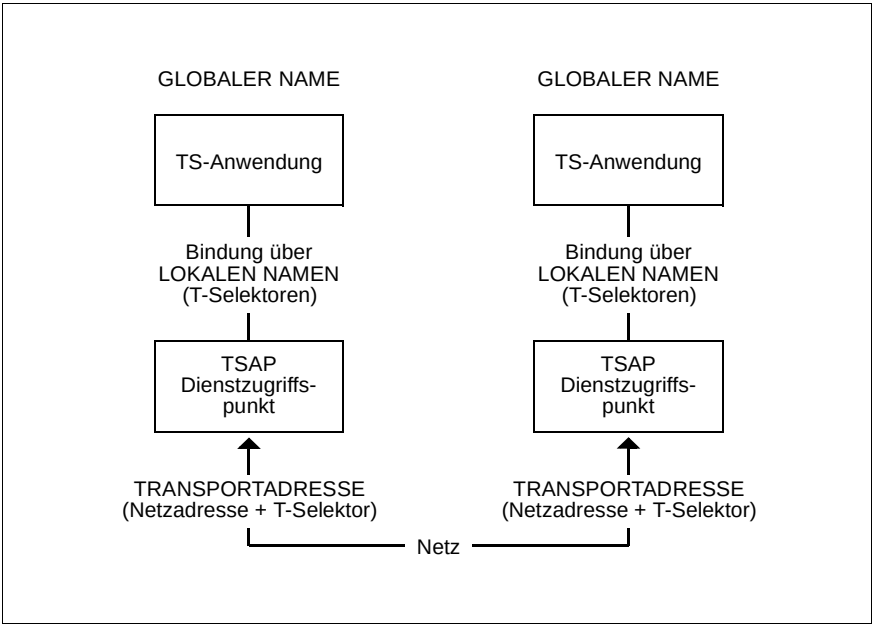

Bild 3: TRANSPORTADRESSE und LOKALER NAME

An der Programmschnittstelle ICMX(L) stehen die Aufrufe *t_getloc()*, *t_getaddr()* und *t_getname()* zur Verfügung, mit denen Sie zu einem GLOBALEN NAMEN den LOKALEN NAMEN bzw. die TRANSPORTADRESSE und zu einer vorgegebenen TRANSPORTADRESSE den zugehörigen GLOBALEN NAMEN abfragen können. Mit diesen Aufrufen können also alle für eine TS-Anwendung an ICMX(L) erforderlichen Informationen aus dem TS-Directory abgefragt werden.

3.2 Struktur einer TS-Anwendung

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

In folgendem Beispiel ist die Struktur eines solchen Programms angedeutet. Die angegebenen Funktionsaufrufe gehören zu der Schnittstelle ICMX(L). Ein Programm, das die Funktionen von ICMX(NEA) anwendet, ist analog strukturiert. Es verwendet statt der t_...()-Aufrufe die entsprechenden x_...()-Aufrufe. Ausnahmen sind die Aufrufe *t_getloc(), t_getaddr() und t_getname()*. Sie können in beiden Programmen verwendet werden.

```
#include <cmx.h>
#include <tnsx.h>
 .
 .
main(argc, argv)
int argc;
char *argv[];
{
    .
    .
    /* 1. Kommunikationsphase */
    t_getloc();             /* Ermittlung LOKALER NAME */
    t_attach();             /* Anmelden bei CMX */
    /* 2. Kommunikationsphase */
    t_getaddr();            /* Ermitteln TRANSPORTADRESSE */
                            /* des Partners */
    t_conrq();              /* Verbindung aufbauen */
    .
    .
    t_concf();      /* Uebernahme Verbindungsbestaetigung */
    /* 3. Kommunikationphase */
    t_datarq();     /* Daten an den Partner senden */
    .
    .
    t_datain();     /* Daten vom Partner empfangen */
    .
    .
    t_disrq();      /* Verbindung abbauen */
    t_detach();     /* Abmelden bei CMX */
    .
    .
    exit();
}
```

Include-Dateien

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

Soll die TS-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, muss sie zusätzlich eine Include-Anweisung für die Datei <*neabx.h*> enthalten. In <*neabx.h*> sind alle zusätzlichen Parameter definiert, die für den Migrationsservice benötigt werden.

Alle diese Dateien stehen im Dateiverzeichnis */usr/include*.

Erlaubte Reihenfolge beim Aufruf der CMX-Funktionen

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

– 1. Kommunikationsphase: Die TS-Anwendung muss sich bei CMX anmelden. Erst wenn CMX die TS-Anwendung kennt, kann diese die Leistungen von CMX nutzen. Die Abläufe in dieser Kommunikationsphase sind in Kapitel 5 beschrieben.

– 2. Kommunikationsphase: In dieser Phase stellt die TS-Anwendung die Verbindung zu ihrem Kommunikationspartner her. Bei dem Verbindungsaufbau müssen sich die beiden Partner einigen, wie der folgende Datenaustausch aussehen soll und welche Form die Daten haben. Beide Partner legen z. B. fest, ob sie auch Vorrangdaten austauschen wollen. Die Abläufe in dieser Kommunikationsphase sind in Kapitel 6 beschrieben.

– 3. Kommunikationsphase: In der dritten Phase werden die Daten zwischen den Partnern ausgetauscht. Beide Kommunikationspartner können Daten senden und empfangen. Die Abläufe in dieser Kommunikationsphase sind in Kapitel 7 beschrieben.

Somit ist für ein TS-Anwendungsprogramm die Reihenfolge festgelegt, in der die CMX-Funktionen aufgerufen werden können. Darüberhinaus muss man beachten, dass einige Aufrufe erst erfolgen dürfen, wenn bestimmte Antworten vom jeweiligen Kommunikationspartner eingetroffen sind und von der TS-Anwendung entgegengenommen wurden (Ereignisse; siehe Abschnitt „Ereignisse entgegennehmen“ auf Seite 37).

Man kann sagen, dass die TS-Anwendung im Laufe der Kommunikation verschiedene Zustände einnimmt. In jeder Kommunikationsphase sind mehrere Zustände möglich. Es sind nur bestimmte Übergänge zwischen den Zuständen innerhalb einer Phase und bestimmte Übergänge zwischen Zuständen verschiedener Phasen möglich.

Eine TS-Anwendung kann nur von einem Zustand in den nächsten übergehen, indem sie bestimmte CMX-Funktionen aufruft oder wenn für sie bestimmte Ereignisse aus dem Netz eintreffen.

In Abschnitt „Zustände von TS-Anwendungen und Zustandsübergänge“ auf Seite 97 und „Zustandsautomaten“ auf Seite 233 sind die möglichen Zustände und die möglichen Zustandsübergänge in Diagrammen dargestellt. Diese Diagramme sollen das Erstellen von TS-Anwendungsprogrammen erleichtern.

Kommunikation einer TS-Anwendung über ICMX(L) und ICMX(NEA)

Sie können in einem TS-Anwendungsprogramm innerhalb einer Kommunikationsphase die Aufrufe von ICMX(L) und ICMX(NEA) nicht mischen. Ausnahmen sind die ICMX(L)-Funktionen *t_getaddr(), t_getloc()* und *t_getname()* zur Abfrage von Namen und Adressen aus dem TS-Directory. Sie können auch von TS-Anwendungen verwendet werden, die den Migrationsservice nutzen. Bitte beachten Sie, dass eine TS-Anwendung zu einer Zeit nur über eine der Schnittstellen ICMX(L) oder ICMX(NEA) kommunizieren kann. Will ein Prozess einer TS-Anwendung gleichzeitig in beiden Modi kommunizieren, so muss er sich als zwei verschiedene TS-Anwendungen, d. h. mit zwei verschiedenen LOKALEN NAMEN, bei CMX anmelden.

Verständigung über die Form der übertragenen Daten

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

Parameterübergabe und Speicherbereitstellung

In TS-Anwendungen werden die Parameter zu den CMX-Funktionen als Werte oder Zeiger übergeben, für Optionen sind Unions definiert. Alle Strukturen sind in den Include-Dateien vereinbart.

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()* (siehe Beschreibung des C-Entwicklungssystems). 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.

3.3 Übersetzen und Binden von TS-Anwendungsprogrammen

Nachdem ein C-Programm *prog.c* einer TS-Anwendung editiert ist, muss es übersetzt (compiliert) und die CMX-Funktionen aus der CMX-Bibliothek *libcmx.so* ins Programm eingebunden werden. Dazu benötigen Sie das C-Entwicklungssystem. Der C-Compiler mit eingeschlossener Bindephase wird wie folgt aufgerufen:

```
cc –o prog prog.c ... –lcmx –lsocket –lnsl
```

Eventuelle Abweichungen von dieser Syntax sind in der Freigabemitteilung beschrieben.

3.4 TS-Anwendungen, Prozesse, Verbindungen

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

3.4.1 TS-Anwendungen und Prozesse

Im einfachsten Fall ist eine TS-Anwendung ein einziger Prozess. Zur Strukturierung einer TS-Anwendung gibt es aber weitere Möglichkeiten.

Eine TS-Anwendung kann mit mehreren Prozessen arbeiten. Die Prozesse müssen nicht miteinander verwandt sein. Jeder einzelne Prozess einer TS-Anwendung muss sich bei CMX anmelden. Prozesse gehören zur selben TS-Anwendung, wenn sie sich mit demselben LOKALEN NAMEN bei CMX angemeldet haben. Der erste Prozess, der sich anmeldet, erzeugt die TS-Anwendung.

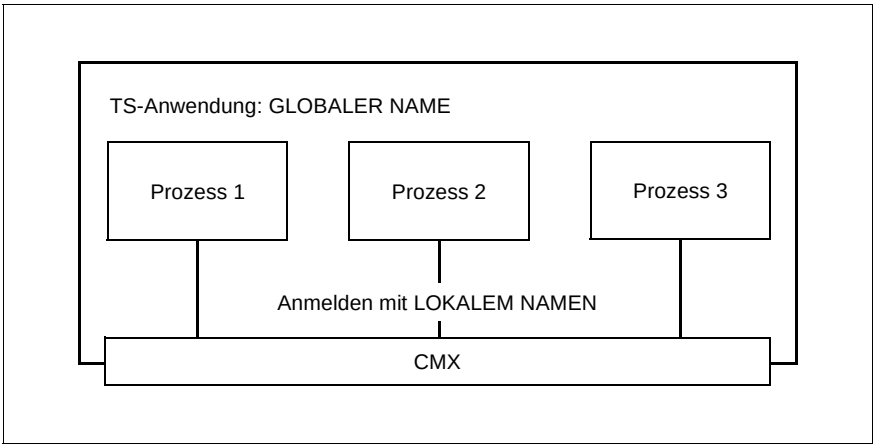

Bild 4: Eine TS-Anwendung - mehrere Prozesse

Andererseits kann ein Prozess mehrere TS-Anwendungen steuern. Dazu meldet man den Prozess mit verschiedenen LOKALEN NAMEN bei CMX an.

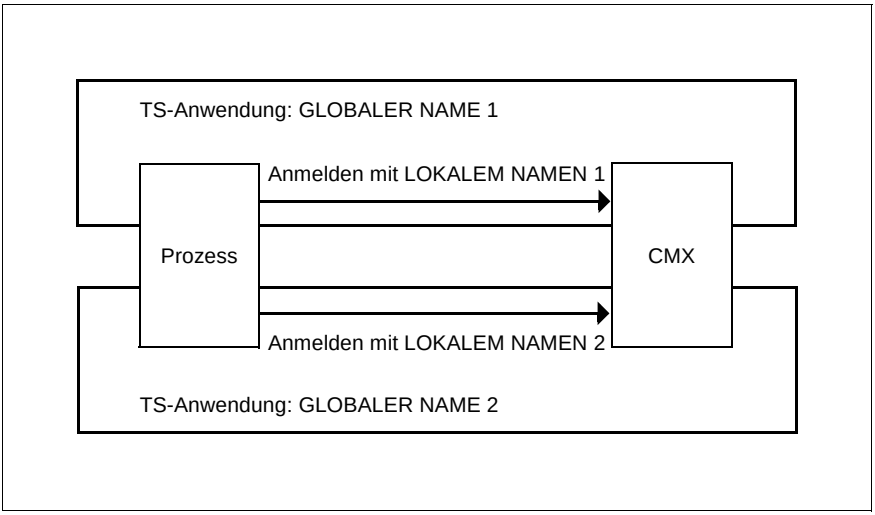

Bild 5: Ein Prozess - mehrere TS-Anwendungen

Der Prozess unterscheidet die verschiedenen TS-Anwendungen, die er steuert, durch die verschiedenen LOKALEN NAMEN oder eine frei wählbare Benutzerreferenz.

3.4.2 Verbindungen und Prozesse

Die Prozesse einer TS-Anwendung können unabhängig voneinander Verbindungen zu anderen TS-Anwendungen aufbauen. Dabei kann ein einzelner Prozess gleichzeitig mehrere Verbindungen halten. Die Verbindungen können auch zu verschiedenen TS-Anwendungen gehören, wenn der Prozess in mehreren TS-Anwendungen angemeldet ist. Beim Verbindungsaufbau wird ein Transportendpunkt (TCEP - Transport Connection Endpoint) für jede Verbindung eingerichtet. Ein Prozess kann also mehrere TCEP bedienen. Ein TCEP kann jedoch nicht gleichzeitig mehreren Prozessen zugeordnet sein. Jede Verbindung ist zu jedem Zeitpunkt **genau** einem Prozess zugegeordnet. Sie ist **nicht** "vererbbar" durch *fork()*.

Jede Verbindung erhält von CMX eine Identifikation, die Transportreferenz. Damit allein kann der Prozess eine Verbindung gezielt ansprechen.

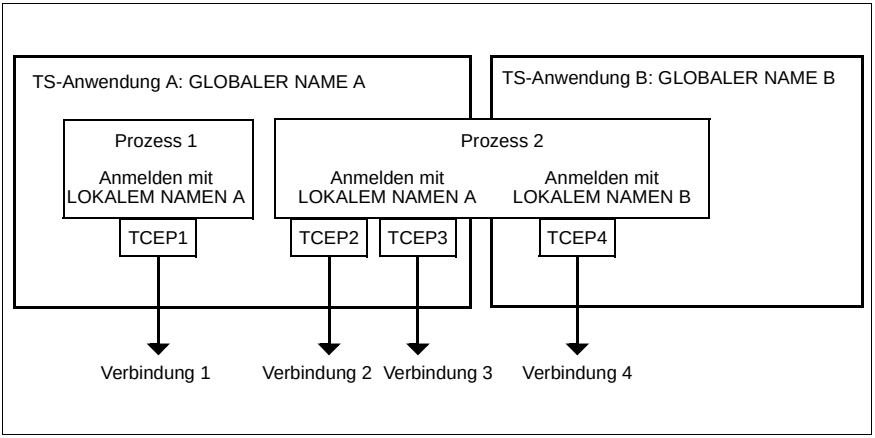

Bild 6: Verbindungen und Prozesse

Ein Prozess kann aber eine Verbindung zu einem anderen Prozess, der sich in derselben TS-Anwendung angemeldet hat, umlenken. Die Verbindung ist dann dem Prozess, der sie abgegeben hat, nicht mehr bekannt. 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. Im Bild oben kann z. B. der Prozess 2 die Verbindung 2 oder 3 an den Prozess 1 umlenken.

3.5 Threads and Multithreading (mehrfädige Anwendungen)

Ein Thread (wörtlich übersetzt "Faden") ist ein sequenziell ablaufender Teil eines Programms. Ein rein sequenzielles Programm nennt man auch single-threaded ("einfädig"). Bestehen die Programme selbst aus mehreren unabhängigen Abschnitten, die gleichzeitig ablaufen können, heißen sie multithreaded.

Ein Multithreading(MT)-Betriebssystem bietet die Möglichkeit, verschiedene Teile innerhalb eines einzigen Prozesses parallel ablaufen zu lassen. Auf Einzelprozessor-Maschinen laufen solche Threads pseudoparallel ab, auf Multiprozessor-Maschinen, hingegen, können sie „echt“ parallel ablaufen.

CMX V6.0 liefert eine Bibliothek für mehrfädige Anwendungen gemäß POSIX 1003.1c bzw. ISO/IEC 9945-11. Hierfür wurden Funktionsaufrufe der Programmierschnittstelle ICMX(L) multithread-fähig gemacht.

Die Programmierschnittstelle ICMX(NEA) wird nicht in einer multithread-fähigen Version angeboten.

Verbindungen und Prozesse

Jeder Thread verwaltet seine eigenen Verbindungen und kann nicht auf Verbindungen zugreifen, die von einem anderen Thread gehalten werden. Verbindungen können, sowohl wie bisher zu einem anderen Prozess (Thread im anderen Prozess), als auch zu einem anderen Thread im gleichen Prozess mit Hilfe der Funktionen *t_redrq* und *t_redin* umgelenkt werden.

Das vorliegende Handbuch beschreibt primär CMX-Anwendungen, die auf dem konventionellen UNIX-Prozessmodell basieren. Bei mehrfädigen Anwendungen treffen die im vorliegenden Handbuch für Prozesse gemachten Aussagen auf Threads zu. Bei der Erstellung von multithreaded CMX-Anwendungen sind insbesondere die Funktionsaufrufe *t_redin* und *t_redrq* zu beachten (siehe Abschnitt „t_redin - Umgelenkte Verbindung annehmen (redirection indication)“ auf Seite 190 und Abschnitt „t_redrq - Verbindung umlenken (redirection request)“ auf Seite 194).

Beispiel 1: TS-Anwendung mit einem Prozess und mehreren Threads

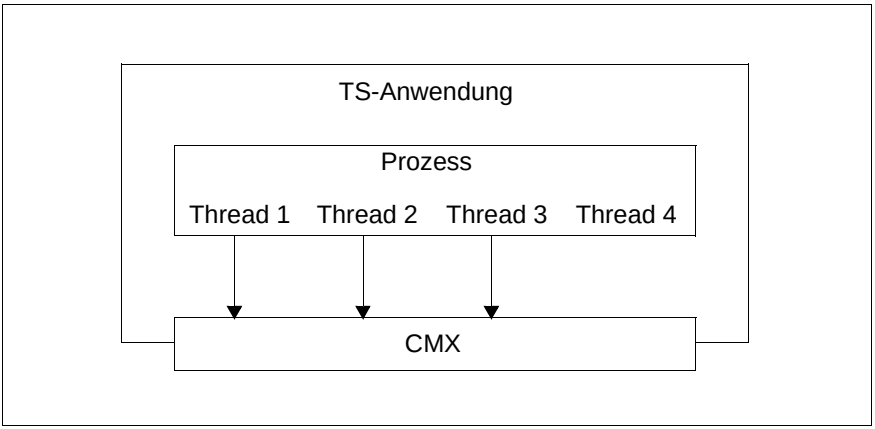

Bild 7: Struktur einer multithreaded TS-Anwendung mit einem Prozess

Der Prozess enthält vier Threads: Thread 1, 2 und 3 sind jeweils bei CMX angemeldet, Thread 4 ist nicht bei CMX angemeldet.

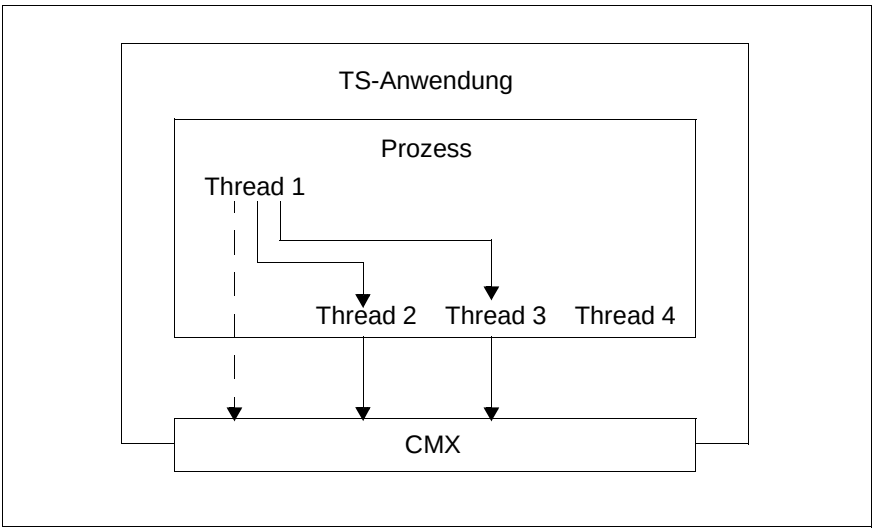

Bild 8: Verbindungsumlenkung innerhalb eines Prozesses mit mehreren Threads

Thread 1 fungiert in diesem Beispiel als Verteiler-Thread. Er ist bei CMX angemeldet und leitet ankommende Verbindungsaufbauwünsche an andere Threads innerhalb des Prozesses um. Diese Threads führen unabhängig voneinander die Datentransfer-Phase durch. Thread 1 könnte jedoch auch selbst zusätzliche Verbindungen aufbauen.

Beispiel 2: TS-Anwendung mit mehreren Prozessen und jeweils mehreren Threads

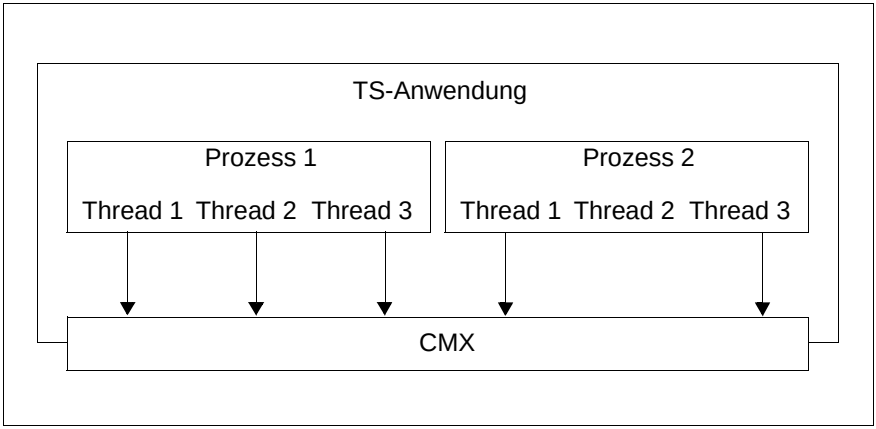

Bild 9: Struktur einer multithreaded TS-Anwendung mit mehreren Prozessen

Die Anwendung arbeitet mit zwei Prozessen, von denen jeder mehrere Threads enthält. Bis auf Thread 2 im Prozess 2 haben sich alle Threads bei CMX angemeldet.

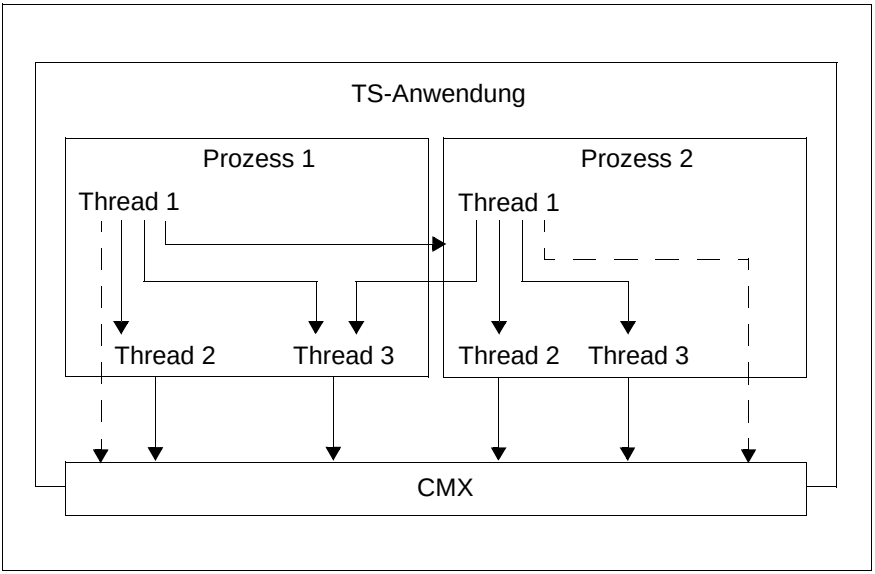

Bild 10: Verbindungsumlenkung zwischen mehreren Prozessen mit mehreren Threads

Die Threads 1 in den Prozessen 1 und 2 fungieren in diesem Beispiel als Verteiler-Threads. Sie sind bei CMX angemeldet und leiten ankommende Verbindungsaufbauwünsche sowohl an andere Threads innerhalb des eigenen Prozesses als auch zu einem Thread im jeweils anderen Prozess um. Sie können Verbindungsaufbauwünsche aber auch selbst bearbeiten.

Include-Dateien

Für die POSIX-Threads sind die Include-Dateien *<pthread.h>*, *<errno.h>*, *<limits.h>*, *<signal.h>*, *<types.h>* und *<unist.h>* erforderlich. Sie befinden sich im Verzeichnis */usr/include*.

CMX-Bibliotheksfunktionen

t_attach()

Jeder Thread, der CMX verwenden will, muss sich mit *t_attach()* bei CMX anmelden. Alle weiteren verbindungsspezifischen CMX-Aufrufe wie *t_datarq()* können nur aus dem Thread erfolgen, der sich mit *t_attach()* angemeldet und mit *t_conrq()*, *t_conin()* oder *t_redin()* eine Verbindung aufgebaut bzw. erhalten hat.
Die CMX-Verbindungen der einzelnen Threads sind strikt voneinander getrennt. Ein Thread kann nicht auf die CMX-Verbindungen eines anderen Threads zugreifen.

t_redin(), t_redrq()

Diese beiden Funktionen verwenden die Prozess-ID, um den Prozess zu identifizieren, zu dem eine Verbindung umgelenkt werden soll, bzw. von dem die Verbindung erhalten wurde. Die Thread-ID muss auch mit angegeben werden, wenn die Verbindung zu einem bestimmten Thread im Empfängerprozess umgelenkt werden soll.

Mögliche Fälle einer Verbindungsumlenkung:

- an einen beliebigen anderen Thread im selben Prozess
- an einen bestimmten anderen Thread im selben Prozess
- an einen beliebigen Thread in einem anderen Prozess
- an einen bestimmten Thread in einem anderen Prozess

Bei Umlenkung zu einem beliebigen Thread wählt CMX diesen Thread aus; es müssen folgenden Merkmale erfüllt sein:

- Er darf Verbindungen durch Umlenkung entgegennehmen (T_REDIN beim *t_attach()* gesetzt).
- Er hat sich bei derselben TS-Applikation angemeldet (LOKALER NAME bei *t_attach()*)
- Sein Verbindungslimit ist noch nicht erschöpft (Parameter *t_conlim* bei *t_attach()*).

t_setopt()

Die Funktion *t_setopt()* verändert nur thread-spezifische Daten. Der Trace kann für den Thread ein- und ausgeschaltet werden. Beim Einschalten wird der Trace-Umfang mit den Optionen -s, -S und -D festgelegt. Parameter, die den gesamten Prozess betreffen, können damit nicht verändert werden.

Übersetzen und Binden

- Um strikte POSIX 1003.1c-Kompatibilität zu gewährleisten, verwenden Sie beim Kompilieren von MT-Programmen den Schalter _POSIX_C_SOURCE=199506L.
- Um multithreaded Code zu kompilieren, setzen Sie den Schalter _REENTRANT.
- Schalten Sie mit der Compile-Option -mt alle Optionen ein, die bei Multithreading gebraucht werden.

Zum Binden können Sie weiterhin den Solaris-Standard-Binder verwenden.

- Binden Sie mehrfädige Anwendungen mit der Bibliothek *libpthreadcmx.so* (Option -lpthreadcmx) anstelle der Bibliothek *libcmx.so* (Option -lcmx).
- Binden Sie Programme mit -lpthread, was den Zugriff auf Definitionen in <*pthread.h*> zur Folge hat: Geben Sie dazu bei der Compile-Anweisung -lpthread als letzten Schalter an.
- Binden Sie beim expliziten Linken *libpthread.so* vor *libc.so* ein (*libc* hat vordefinierte libpthread stubs, d.h. Dummy-Funktionen).
- Platzieren Sie den Schalter -lpthread bei der Link-Anweisung (ld) vor dem Schalter -lc.

Beispiele:

```
cc -mt [flags] file ... -lpthreadcmx -lpthread -lc
cc -mt [flags] file ... -lpthreadcmx -D_POSIX_C_SOURCE=199506L \
                    -D_EXTENSIONS_ -lpthread -lc
```

Weitere Details zum Kompilieren und Binden einer multithreaded Applikation finden sich im „Multithreaded Programming Guide“ von Sun Microsystems.

Der Schalter -mt entspricht -D_REENTRANT plus -lthread.
Non-threaded und single-threaded Applikationen werden ohne die Flags _REENTRANT, _POSIX_C_SOURCE und __EXTENSIONS__ (bzw. mt) kompiliert.

Signale

CMX single-threaded unterstützt ein beliebiges Signal (z. B. SIGIO), um die Applikation über das Vorhandensein eines Ereignisses zu informieren. Damit kann eine Applikation z.B. blockierend von *stdin* lesen und wird unterbrochen, sobald ein CMX-Ereignis vorliegt. Da nicht sichergestellt ist, dass ankommende Signale dem richtigen Thread zugestellt werden, wird diese Funktionalität in der multithreaded CMX-Bibliothek nicht unterstützt.

Weitere Hinweise

Thread-ID

Die Thread-ID ist vom Datentyp pthread_t, nicht vom Typ Integer.

Variable errno

Die Variable *errno* wird durch die Verwendung der Include-Datei *<errno.h>* thread-spezifisch.

Thread-Erzeugung, -Beendigung

CMX-Funktionen erzeugen weder Threads, noch beenden sie welche.

Detach-Status

Mit den Funktionen *pthread_attr_setdetachstate()* bzw. *pthread_attr_getdetachstate()* bestimmen Sie, ob die Ressourcen des Threads erneut verwendet werden.

Bei CMX angemeldete Threads dürfen nur mit *t_detach()* abgemeldet werden. Wenn Sie diese mit Thread-Bibliotheksfunktionen wie z.B. *pthread_detach()*, *pthread_exit()* oder *pthread_cancel()* beenden, können die thread-spezifischen CMX-Ressourcen verloren gehen.

Stack Handling

Standardgröße ist 1 Mbyte.
Sie können jedoch mittels der Threadfunktionen *pthread_attr_setstacksize()* bzw. *pthread_attr_setstackaddr()* die Stack Size festlegen. CMX selbst hat keinen darüber hinaus gehenden Bedarf. Als Minimum sollten Sie für CMX 32 kByte einplanen.

Bibliotheks-Trace

In der Ausgabe des Bibliotheks-Trace wird zusätzlich die Thread-ID aufgeführt. Dadurch ist auch eine thread-spezifische Nachbehandlung der durch cmxl aufbereiteten ASCII-Trace-Daten möglich. Im Vorspann der ASCII-Trace-Datei wird ausgegeben, ob sie von einer single- oder multithreaded Applikation erzeugt wurde. Siehe auch Handbuch „CMX, Betrieb und Administration“ [1].

4 Ereignisverarbeitung und Fehlerbehandlung

4.1 Ereignisse entgegennehmen

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

Beispiele für solche Ereignisse sind:

– der Verbindungswunsch eines Kommunikationspartners (der "rufenden Anwendung")

– die Ankunft von Daten auf einer bestehenden Verbindung

– Fluss-Regelungsereignisse (Sendesperre gesetzt bzw. aufgehoben)

– der Verbindungsabbau durch den Kommunikationspartner oder CMX

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

Mit dem Aufruf *t_callback()* kann eine Routine an CMX übergeben werden, welche anstelle des CMX-internen „Warten/Prüfen auf Ereignisse" aufgerufen wird. Das Programm wartet in dieser Routine auf CMX- und auf programmspezifische Ereignisse.

Die Funktionen von CMX sind so ausgelegt, dass die TS-Anwendung nach Absetzen eines Aufrufs auf die eventuelle Antwort aus dem Netz warten kann, aber nicht muss. Sie hat drei Möglichkeiten Ereignisse zu verarbeiten:

– Synchrone Verarbeitung

– Asynchrone Verarbeitung

– Ereignisverarbeitung im Programm

Synchrone Verarbeitung

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

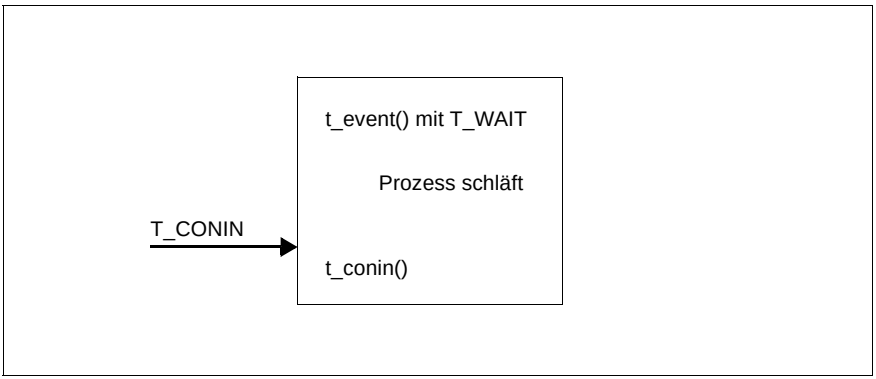

Bild 11: Synchrone Verarbeitung

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

Beim Aufruf von *t_event()* kann man auch die Wartezeit beschränken. Man gibt einfach an, wie lange der Prozess auf ein Ereignis warten soll. Trifft in dieser Zeit kein Ereignis ein, setzt CMX den Prozess mit T_NOEVENT fort.

Asynchrone Verarbeitung

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

Es ist aber nicht sinnvoll, nur *t_event()* in einer dauernden Schleife laufen zu lassen, besser sollten Sie dann die synchrone Ereignisverarbeitung (*cmode* = T_WAIT) verwenden, und den Prozess, wenn nötig, periodisch mit *alarm()* aufwecken.

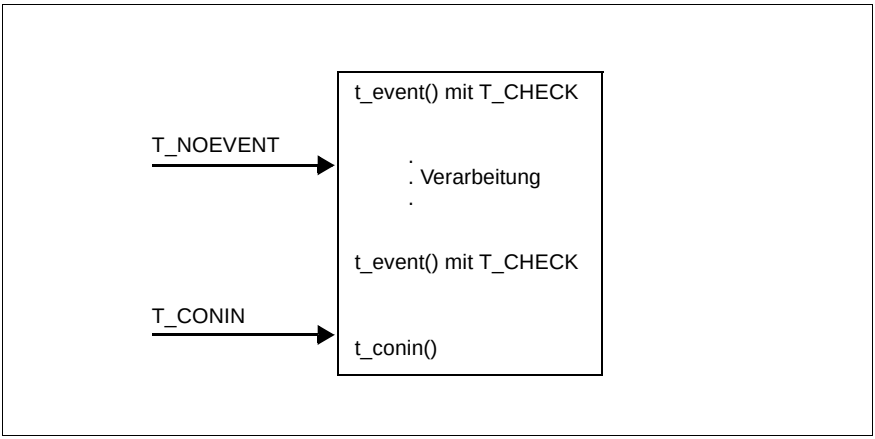

Bild 12: Asynchrone Verarbeitung

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 TS-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen Sie die Ereignisse mit dem Aufruf *x_event()* abfragen.

Ereignisverarbeitung im Programm

Mit dem Aufruf *t_callback()* wird eine eigene Rückrufroutine eingehängt. Diese Routine wird anstelle des CMX-internen „Warten/Prüfen auf Ereignisse" während des *t_event()* aufgerufen. In der callback-Routine muss das Programm auf CMX-Ereignisse warten/prüfen und kann auch auf eigene Ereignisse warten/prüfen.

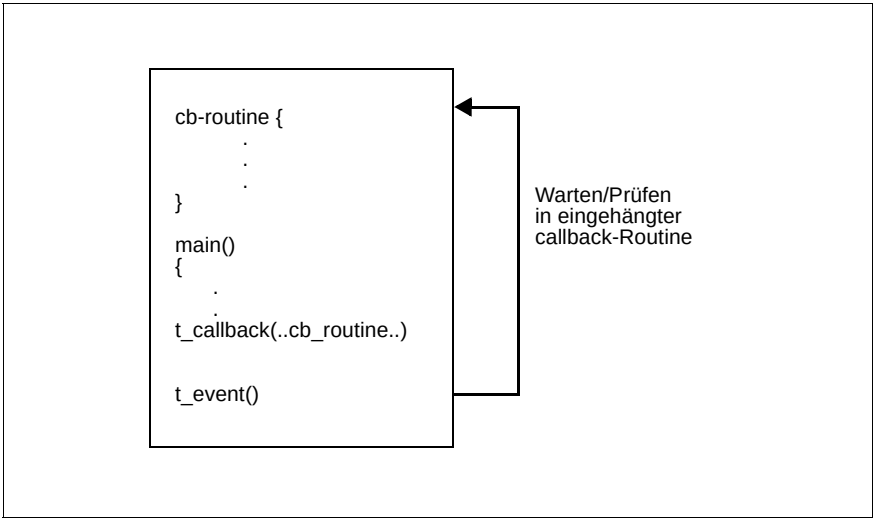

Bild 13: Einhängen und Anspringen einer callback-Routine

4.2 Fehlerbehandlung

4.2.1 Funktionen zur Fehlerabfrage

Ein fehlerhaft abgelaufener Funktionsaufruf kehrt immer mit einer globalen Fehleranzeige zurück. Einen detaillierteren Wert erhält man durch den Aufruf einer Fehlerabfragefunktion. Die folgende Tabelle gibt an welche Anzeige und welche Funktion zu den einzelnen CMX-Programmschnittstellen gehören.

Schnittstelle	**Funktions-aufrufe**	**globale Fehler-anzeige**	**Fehlerabfrage-funktion**
ICMX(L)	t_....	T_ERROR	t_error()
ICMX(NEA)	x_....	X_ERROR	x_error()

Tabelle 4: Funktionen zur Fehlerabfrage

Die von *t_error()* oder *x_error()* gelieferten Werte sind hexadezimal codiert.

4.2.2 Aufbau der CMX-Fehlermeldungen

Jede Fehlermeldung an ICMX(L) und ICMX(NEA) wird übergeben in der Form: 0x%x.

%x ist dabei ein 16 bit langer Fehlercode. Der Fehlercode ist wie folgt aufgebaut.

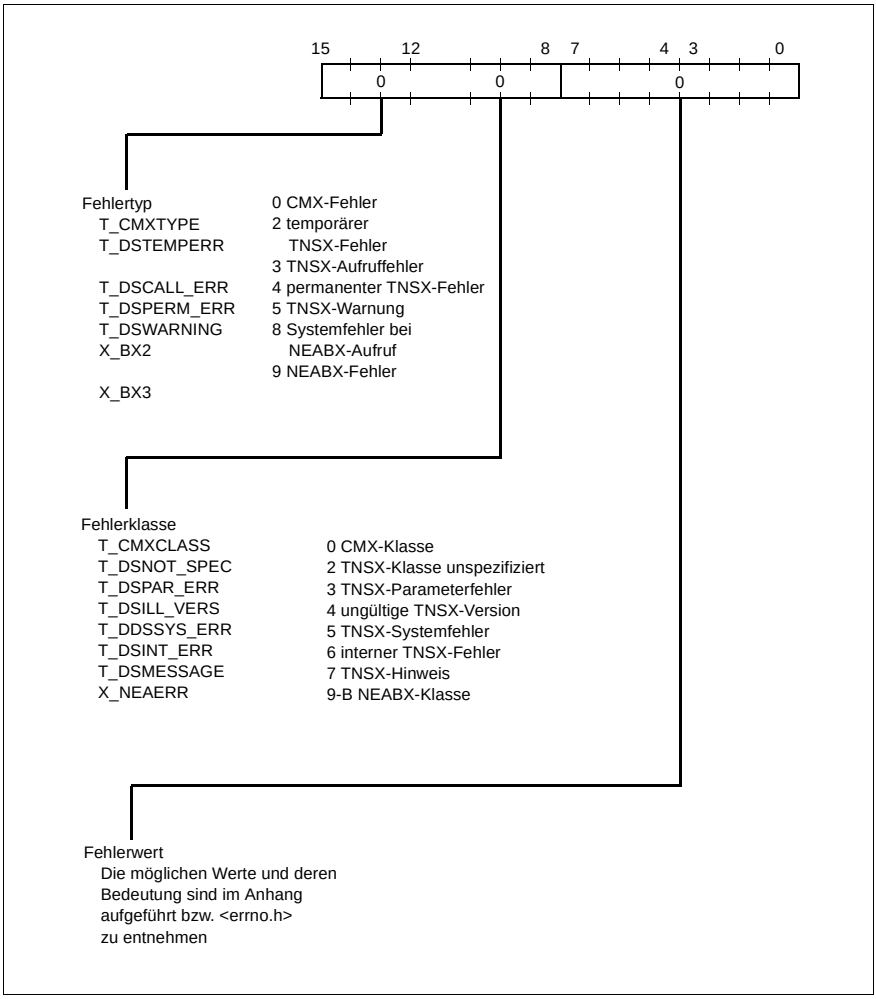

Bild 14: Aufbau des Fehlercodes

Die Fehlermeldung ist von links (Bit 15) her auszuwerten.

Im Anhang sind die Fehlerwerte zu den einzelnen Funktionen von ICMX(L) und ICMX(NEA) aufgelistet, sofern es sich um in CMX erzeugte Fehler handelt. Andere Fehlerwerte sind <*errno.h*> zu entnehmen.

4.2.3 Decodieren der Fehlermeldungen

An den Schnittstellen ICMX(L) und ICMX(NEA) stehen zusätzlich zu den Diagnosefunktionen *t_error()* und *x_error()* Funktionsaufrufe zur Verfügung, mit denen man die Fehlercodes in Klartextdarstellung übersetzen kann.

Mit dem Programm *cmxdec* können Sie ICMX-Fehlermeldungen und Verbindungsabbaugründe auf Kommanoebene decodieren.

Verbindungsabbaugründe sind die Werte, die bei einem der Aufrufe *t_disin()* oder *x_disin()* im Parameter *reason* zurückgeliefert werden. Sie geben an, warum eine Verbindung abgebaut bzw. abgelehnt wurde.

Der Fehlercode bzw. der Wert von *reason* wird von *cmxdec* entschlüsselt. Der in der entsprechenden Include-Datei definierte symbolische Wert wird auf *stderr* ausgegeben. Falls ein entsprechender Messagekatalog existiert, wird zusätzlich ein erläuternder Text ausgegeben. Das Programm *cmxdec* ist in den Handbüchern „CMX, Betrieb und Administration“ [1] und [2] beschrieben.

5 An- und Abmelden bei CMX

Eine TS-Anwendung entsteht, sobald sich ein Prozess mit dem LOKALEN NAMEN der TS-Anwendung bei CMX anmeldet. Jeder weitere Prozess, der in dieser TS-Anwendung arbeiten will, muss sich ebenfalls für diese TS-Anwendung, d. h. mit ihrem LOKALEN NAMEN, bei CMX anmelden.

Bevor sich ein Prozess beendet, muss er sich bei CMX abmelden. Hat sich der letzte Prozess einer TS-Anwendung bei CMX abgemeldet, so existiert diese TS-Anwendung für CMX nicht mehr.

5.1 Anmelden bei CMX

Die Anmeldung eines Prozesses bei CMX über die Programmschnittstelle ICMX(L) erfolgt mit dem Aufruf *t_attach()*.

Bei der Anmeldung muss der Prozess den LOKALEN NAMEN der TS-Anwendung übergeben, für die er sich bei CMX anmelden will. Den LOKALEN NAMEN muss der Prozess vor der Anmeldung, d. h. vor dem Aufruf von *t_attach()*, aus dem TS-Directory lesen. Dazu ruft er die ICMX(L)-Funktion *t_getloc()* auf. Er übergibt *t_getloc()* als Parameter den GLOBALEN NAMEN der TS-Anwendung, für die er sich anmelden will. *t_getloc()* liefert den Zeiger auf eine Struktur zurück, in der der LOKALE NAME steht. Dieser Zeiger wird beim Aufruf *t_attach()* als Parameter übergeben.

Der Aufruf *t_getloc()* muss also vor dem Aufruf *t_attach()* erfolgen.

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

Jeder Prozess einer TS-Anwendung legt bei der Anmeldung für sich fest, ob:

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

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

– er Verbindungen annimmt, die ein anderer Prozess der gleichen TS-Anwendung an ihn übergeben will (Verbindungsumlenkung annehmen). Ein Prozess der gleichen TS-Anwendung ist ein Prozess, der sich mit demselben LOKALEN NAMEN bei CMX angemeldet hat.

Ein Prozess kann sich für alle drei dieser Möglichkeiten bei CMX anmelden, oder nur für eine oder zwei davon.

Derselbe Prozess kann sich auch für mehrere verschiedene TS-Anwendungen anmelden. Dazu ruft er *t_attach()* und *t_getloc()* für jede dieser TS-Anwendungen auf.

CMX nimmt für eine TS-Anwendung Verbindungsanforderungen von entfernten TS-Anwendungen entgegen, sobald sich ein Prozess der TS-Anwendung zum passiven Verbindungsaufbau bei CMX angemeldet hat. Ankommende Verbindungswünsche gibt CMX zunächst an den Prozess weiter, der sich als Erster in dieser TS-Anwendung zum passiven Verbindungsaufbau angemeldet hat.

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

Anmelden über ICMX(NEA)

Die Anmeldung einer TS-Anwendung an der Programmschnittstelle ICMX(NEA) läuft analog ab. Es muss lediglich statt der Funktion *t_attach()* die Funktion *x_attach()* verwendet werden. Der LOKALE NAME kann auch hier mit Hilfe von *t_getloc()* aus dem TS-Directory abgefragt werden.

5.2 Abmelden bei CMX

Bevor sich ein Prozess beendet, ruft er *t_detach()* auf. *t_detach()* meldet den Prozess bei CMX für die TS-Anwendung ab. Zuvor müssen aber alle TS-Verbindungen, die dieser Prozess hält, abgebaut werden (siehe Kapitel „Verbindungen verwalten“ auf Seite 49). Tut der Prozess das nicht, baut CMX implizit selbst alle TS-Verbindungen ab. Dies ist aber nur für Ausnahmesituationen vorgesehen, z. B. wenn sich ein Prozess unerwartet frühzeitig beendet.

Sobald sich der letzte Prozess aus einer TS-Anwendung abgemeldet hat, existiert diese TS-Anwendung für CMX nicht mehr. Verbindungsanforderungen von fernen TS-Anwendungen werden für diese TS-Anwendung nicht mehr angenommen.

Abmelden über ICMX(NEA)

Das Abmelden aus einer TS-Anwendung läuft an der Programmschnittstelle ICMX(NEA) analog ab. Es ist lediglich statt der Funktion *t_detach()* die Funktion *x_detach()* zu verwenden.

5.3 Beispiele zur An- und Abmeldung eines Prozesses

5.3.1 Beispiel zum An- und Abmelden an ICMX(L)

Das folgende Programmfragment zeigt den Programmablauf zum Anmelden und Abmelden eines Prozesses an ICMX(L).

Ein Prozess meldet sich für die TS-Anwendung „TestanwendungAKT“ bei CMX an und wieder ab. In der Optionsstruktur *t_opta1* gibt er an, dass er in dieser TS-Anwendung nur aktiv Verbindungen aufbauen will (T_ACTIVE) und gleichzeitig maximal 1 Verbindung halten will.

```
#include        <stdio.h>
#include        <cmx.h>
#include        <tnsx.h>
.
.
#define ERROR   1
 .
 .
struct  t_opta1 t_opta1 = { T_OPTA1, T_ACTIVE, 1 }
                                        /* t_attach () */
 .
 .
 /* Strukturen für Adressierung */

#define MYNAME  "TestanwendungAKT"
char myname[TS_LPN+1] = { MYNAME } ;
struct t_myname t_myname, *p_myname;
   .
   .
/* Aktive Anwendung bei CMX anmelden */

if ((p_myname = t_getloc(myname, NULL)) != NULL)
        t_myname = *p_myname;
else {
   fprintf(stderr, ">>> FEHLER 0x%x bei t_getloc\n",t_error());
   exit(ERROR);
}
if (t_attach(&t_myname, &t_opta1) == T_ERROR) {
    fprintf(stderr, ">>> FEHLER 0x%x bei t_attach\n",t_error());
    exit(ERROR);
}
fprintf(stderr, "Anwendung '%s' angemeldet.\n", myname);
```

```
  .
  .

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

5.3.2 Beispiel zum An- und Abmelden an ICMX(NEA)

Das folgende Programmfragment zeigt den Programmablauf zum Anmelden und Abmelden eines Prozesses an ICMX(NEA).

Ein Prozess meldet sich für die TS-Anwendung "NEAAnwendungAKT" an und wieder ab. In der Optionsstruktur *x_opta1* gibt er an, dass er in dieser TS-Anwendung nur aktiv Verbindungen aufbauen will (X_ACTIVE) und gleichzeitig maximal 1 Verbindung halten will.

```
#include        <stdio.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <neabx.h>
.
.
#define ERROR   1
 .
 .
struct  x_opta1 x_opta1 = { X_OPTA1, X_ACTIVE, 1 };
                                                  /* x_attach () */
 .
 .
 /* Strukturen für Adressierung */

#define MYNAME  "NEAAnwendungAKT"
char myname[TS_LPN+1] = { MYNAME } ;
struct x_myname x_myname, *p_myname;
   .
   .
/* Aktive Anwendung bei ICMX(NEA) anmelden */
if ((p_myname = t_getloc(myname, NULL)) != NULL)
        x_myname = *p_myname;
else {
   fprintf(stderr, ">>> FEHLER 0x%x bei t_getloc\n", t_error());
   exit(ERROR);
```

```
}
if (x_attach(&x_myname, &x_opta1) == X_ERROR) {
   fprintf(stderr, ">>> FEHLER 0x%x bei x_attach\n", x_error());
   exit(ERROR);
}
fprintf(stderr, "Anwendung '%s' angemeldet.\n", myname);
  .
  .
/* TS-Anwendung bei ICMX(NEA) abmelden */

if (x_detach(&x_myname) == X_ERROR)
   fprintf(stderr, ">>> FEHLER 0x%x bei x_detach\n", x_error());
fprintf(stderr, "Anwendung '%s' abgemeldet.\n", myname);
   .
   .
```

6 Verbindungen verwalten

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

Dass in den Diagrammen CMX 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.

6.1 Verbindung aufbauen

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

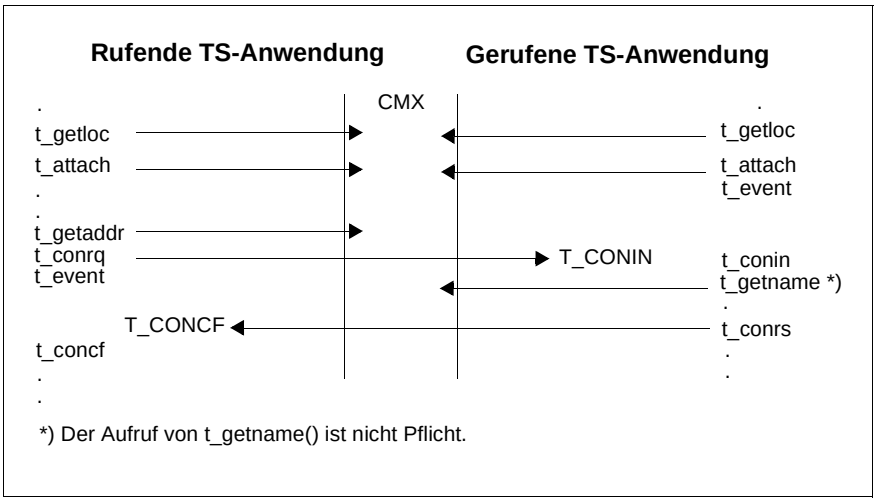

Bild 15: Verbindung aufbauen (ICMX(L))

Ablauf des Verbindungsaufbaus in der rufenden TS-Anwendung

Der Prozess der rufenden TS-Anwendung muss CMX bei der Anmeldung mitteilen, dass er die Absicht hat, eine Verbindung aktiv aufzubauen. Die rufende TS-Anwendung holt sich zuerst ihren LOKALEN NAMEN und meldet sich dann bei CMX an. Danach ermittelt sie die TRANSPORTADRESSE der gerufenen TS-Anwendung und fordert mit *t_conrq()* einen Verbindungsaufbau an. Dann wartet sie mit *t_event()* auf die Bestätigung der gerufenenTS-Anwendung, d.h. auf das TS-Ereignis T_CONCF. Wenn *t_event()* das TS-Ereignis gemeldet hat, stellt die rufende TS-Anwendung die Verbindung mit dem Aufruf *t_concf()* her.

Ablauf des Verbindungsaufbaus in der gerufenen TS-Anwendung

Jeder Prozess der gerufenen TS-Anwendung muss CMX bei der Anmeldung mitteilen, dass er die Absicht hat, eine Verbindung passiv aufzubauen. Die gerufene TS-Anwendung wartet nach der Anmeldung zunächst mit *t_event()* auf ein TS-Ereignis. Das TS-Ereignis T_CONIN zeigt den Verbindungsaufbauwunsch der rufenden TS-Anwendung an. Mit dem Aufruf *t_conin()* nimmt die gerufene TS-Anwendung diesen Verbindungswunsch an. Sie kann dann aus der TRANSPORTADRESSE der rufenden TS-Anwendung deren GLOBALEN NAMEN ermitteln und beantwortet den Verbindungswunsch mit *t_conrs()*.

Benutzerdaten beim Verbindungsaufbau austauschen

Die Aufrufe *t_conin()* (Entgegennehmen der Verbindungsaufbauanzeige) und *t_concf()* (Herstellen der Verbindung) sind nötig, da beide TS-Anwendungen schon beim Verbindungsaufbau Benutzerdaten austauschen können, sofern das Transportsystem diese Option bietet (siehe Abschnitt „Optionen des Systems und des Benutzers“ auf Seite 12).

Bei *t_conrq()* kann die rufende TS-Anwendung Benutzerdaten mitgeben. Das ist eine kleine Datenmenge, die die gerufene TS-Anwendung bei *t_conin()* erhält. Wenn die gerufene TS-Anwendung dann den Verbindungswunsch mit *t_conrs()* beantwortet, kann sie wiederum Informationen mitgeben. Diese erhält die rufende TS-Anwendung bei *t_concf()*.

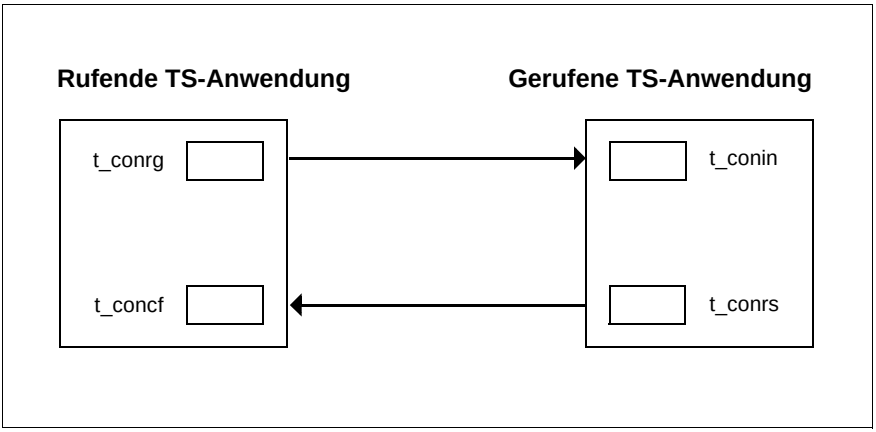

Bild 16: Benutzerdatenaustausch beim Verbindungsaufbau

Verbindungswunsch ablehnen

Die gerufene TS-Anwendung kann den Verbindungswunsch auch ablehnen. Der Ablauf ist derselbe. Das Ereignis T_CONIN muss zunächst mit *t_conin()* angenommen werden, statt des Aufrufes *t_conrs()* ist aber *t_disrq()* zu verwenden (siehe auch Abschnitt „Verbindung abbauen“ auf Seite 55).

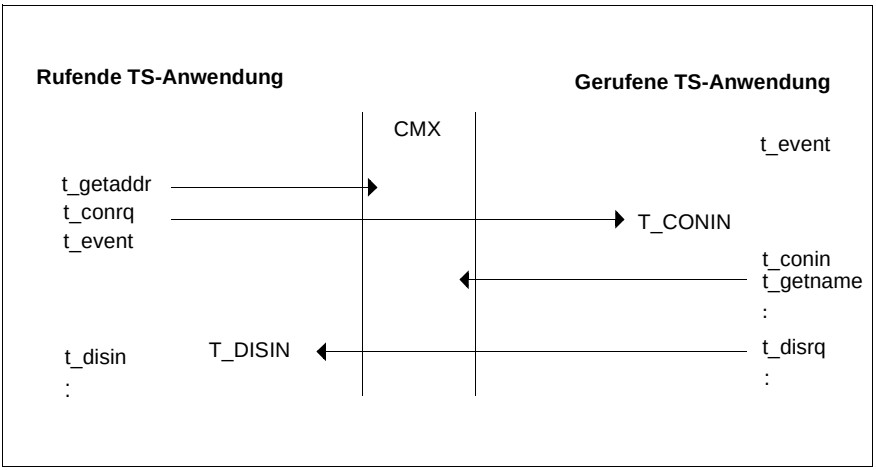

Bild 17: Verbindungswunsch ablehnen

Bemerkungen zum Verbindungsaufbau an ICMX(NEA)

Die Funktionen der Programmschnittstelle ICMX(NEA) rufen intern ICMX(L)-Funktionen auf. Die oben beschriebenen Vorgänge beim Verbindungsaufbau gelten deshalb analog, solange die zu übertragenden Benutzerdaten direkt von CMX übertragen werden können. TS-Anwendungen an ICMX(NEA) müssen in den Benutzerdaten jedoch das NEABV-Protokoll übertragen. Die Länge dieser Benutzerdaten kann die vom Transportsystem zugelassene Benutzerdatenlänge überschreiten. Das hat z. B. zur Folge, dass CMX die beim *x_conrq()* übergebenen Benutzerdaten nicht durch einen Aufruf *t_conrq()* übertragen kann. Der zeitliche Ablauf der daraus resultierenden Vorgänge ist im folgenden Bild dargestellt.

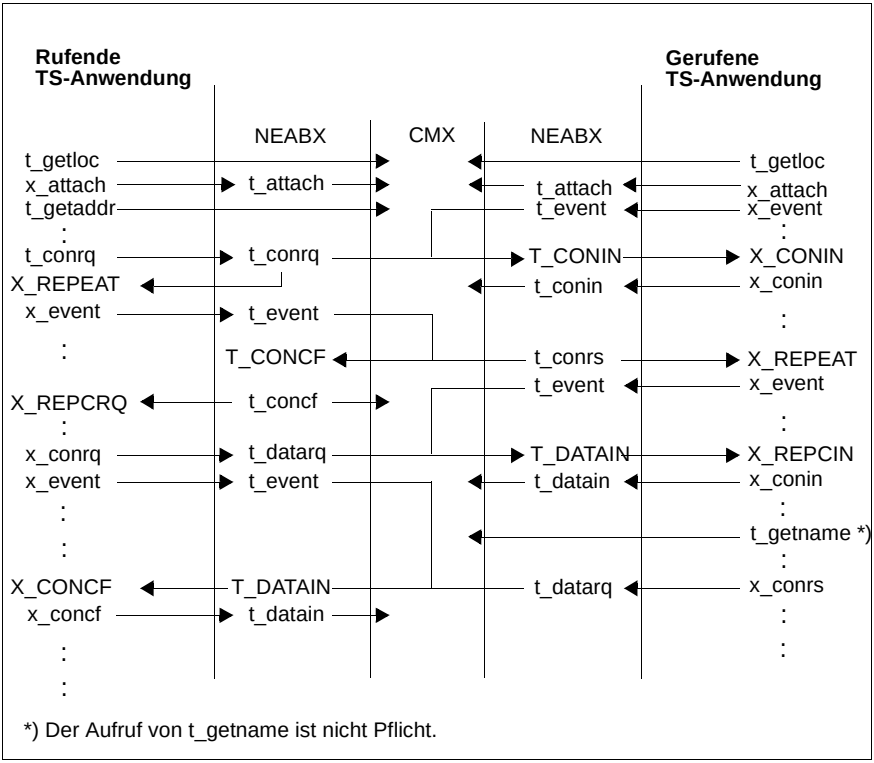

Bild 18: Verbindung aufbauen mit ICMX(NEA)

Die **rufende TS-Anwendung** erhält beim Aufruf von *x_conrq()* das Ergebnis X_REPEAT zurück. Sie muss nun *x_event()* aufrufen und warten bis NEABX ihr das Ereignis X_REPCRQ meldet. Dann muss sie *x_conrq()* mit gleichen Parametern wiederholen. Erst wenn die TS-Anwendung beim Aufruf von *x_conrq()* das Ergebnis T_OK zurückgeliefert bekommt und der folgende Aufruf *x_event()* das Ereignis X_CONCF meldet, kann die TS-Anwendung mit *x_concf()* die Verbindung zur gerufenen TS-Anwendung herstellen.

Die **gerufene TS-Anwendung** ruft nach der Anmeldung *x_event()* auf. Trifft die Verbindungsaufbauanzeige X_CONIN ein, so ruft die gerufene TS-Anwendung *x_conin()* auf. Als Ergebnis wird ihr der Wert X_REPEAT zurückgeliefert. Die TS-Anwendung muss nun erneut *x_event()* aufrufen und auf das Ereignis X_REPCIN warten. Nach dessen Eintreffen nimmt die TS-Anwendung durch die Wiederholung des Aufrufs *x_conin()* die Benutzerdaten entgegen und kann mit *x_conrs()* die Verbindungsanforderung positiv beantworten.

Intern baut CMX bei dieser Form des Verbindungsaufbaus eine Verbindung zum Partner-CMX auf und übermittelt die Benutzerdaten mit Hilfe der ICMX(L)-Aufrufe für die Datenübertragung (*t_datarq(), t_datain()*).

Will die gerufene TS-Anwendung die Verbindung ablehnen, so muss sie anstelle von *x_conrs() x_disrq()* aufrufen.

Vorrangdaten aushandeln

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

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

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

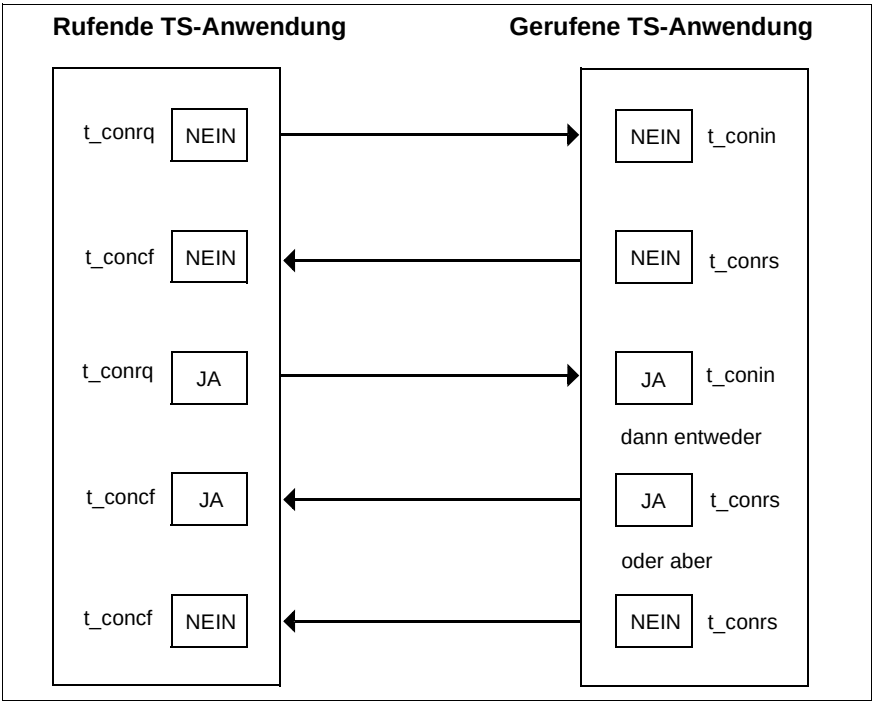

Bild 19: Vorrangdatenverhandlung beim Verbindungsaufbau

Sollen die TS-Anwendungen über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so muss für die o.g. Aufrufe der Präfix x_ anstatt t_ und für die Ereignisse der Präfix X_ anstatt T_ verwendet werden.

6.2 Verbindung abbauen

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

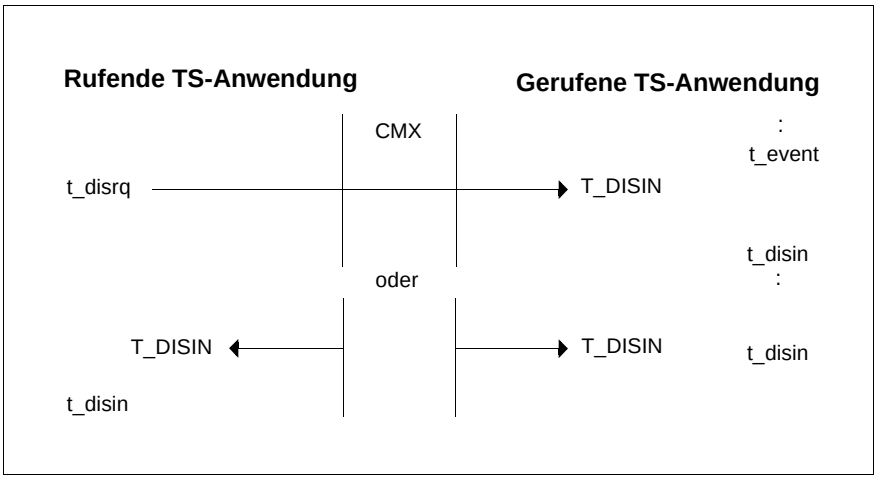

Bild 20: Verbindung abbauen

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

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

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

6.3 Beispiel zum Verbindungsaufbau und -abbau mit ICMX(L)

6.3.1 Beispiele zum Verbindungsaufbau mit ICMX(L)

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

Beispiel 1

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

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

 /* Strukturen für Adressierung */

#define PNAME   "TestanwendungPAS"
char pname[TS_LPN+1] = { PNAME } ;
struct t_partaddr t_partaddr, *p_partaddr;
   .
   .
/* Verbindung aufbauen zum passiven Partner */

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

```
  exit(ERROR);
}

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

             /* Verbindungsabbau durch Partner oder System */

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

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

Beispiel 2

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

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

        case T_CONIN:

          /* Verbindungsaufbauwunsch akzeptieren */

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

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

```
            .

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

6.3.2 Beispiele zum Verbindungsaufbau mit ICMX(NEA)

Beispiel 1 zeigt den Programmaufbau für die rufende TS-Anwendung, Beispiel 2 zeigt den Programmaufbau für die gerufene TS-Anwendung.

Beispiel 1

Die TS-Anwendung baut aktiv eine Verbindung zu der TS-Anwendung „NEAAnwendungPAS“ auf und wieder ab.

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

 /* Strukturen für Adressierung */

#define PNAME   "NEAAnwendungPAS"
char pname[TS_LPN+1] = { PNAME } ;
struct x_partaddr x_partaddr, *p_partaddr;
struct x_optc1 x_optc1 ;
char   *udatap = "Benutzerverbindungsnachricht, länger als 32
                  Zeichen" ;
int     retval ;
char    antwort[X_MSG_SIZE] ;
                    /* Benutzernachricht mit x_concf erhalten */
   .
   .
/* Verbindung aufbauen zum passiven Partner */

if ((p_partaddr = t_getaddr(pname, NULL)) != NULL)
         x_partaddr = *p_partaddr;
else {
      fprintf(stderr, ">>> FEHLER 0x%x bei t_getaddr\n",
      t_error());
      exit(ERROR);
}
x_optc1.x_optnr = X_OPTC3 ;
x_optc1.x_xdata = X_YES ;
```

```
x_optc1.x_timeout = T_NOLIMIT ;
x_optc1.x_prot = X_NEABX ;
x_optc1.x_udatap = udatap ;
x_optc1.x_udatal = strlen(udatap) ;
if ((retval=x_conrq(&tref, (union x_address *)&x_partaddr,
          (union x_address *)&x_myname, &x_optc1)) == X_ERROR) {
     fprintf(stderr, ">>> FEHLER Ox%x bei x_conrq, tref Ox%x\n",
           x_error(), tref);
     exit(ERROR);
}

/* Ereignisgesteuerte Verarbeitung:
 * x_event() synchron (X_WAIT) wartend
 */
for (;;) {
        switch (event = x_event(&tref, X_WAIT, NULL)) {
        case X_REPCRQ :
        if (x_conrq(&tref, (union x_address *)&x_partaddr,
              (union x_address *)&x_myname, &x_optc1) ==
               X_ERROR) {
      fprintf(stderr, ">>> FEHLER Ox%x bei wiederholtem
          x_conrq, tref Ox%x\n", x_error(), tref);
          exit(ERROR);
           }
        break ;
        case X_CONCF:
           /*
            * Verbindungsaufbau gelungen?
            */
           x_optc1.x_udatap = antwort ;
           x_optc1.x_udatal = sizeof(antwort) ;

           if ((retval=x_concf(&tref, &x_optc1)) == X_ERROR) {
                  fprintf(stderr, ">>> FEHLER Ox%x bei x_concf
                  tref Ox%x\n",x_error(), tref);
                  exit(ERROR);
           }
           if ( retval == X_REPEAT )
              break ;
           else
               fprintf(stderr, "Verbindung zu '%s
               aufgebaut.\n", pname);
           .
           .
        case X_REPCCF :
              /*
               * Confirmation wiederholt abholen
               */
```

```
          if (x_concf(&tref, &x_optc1) == X_ERROR) {
      fprintf(stderr, ">>> FEHLER 0x%x bei wiederholtem x_concf
      tref 0x%x\n",x_error(), tref);
                exit(ERROR);
          }
          fprintf(stderr, "Verbindung zu '%s' aufgebaut.\n",
          pname);
          .
          .
        case X_DISIN:

            /* Verbindungsabbau durch Partner oder System */

            if (x_disin(&tref, &reason, NULL) == X_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei x_disin
                tref 0x%x\n",x_error(), tref);
                exit(ERROR);
            }
            fprintf(stderr, "Verbindungsabbau erhalten, tref
            0x%x,reason %d\n", tref, reason);
            :
        }
}

 /* Verbindungsabbau */

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

Beispiel 2

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

```
#include        <stdio.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <neabx.h>
    .
    .
#define ERROR   1
    .
    .
int     tref;                   /* Transportreferenz */
int     reason;                 /* Grund für Verbindungsabbau */
/*
 * Strukturen für Adressierung
 */
struct x_myname x_myname, *p_myname;
struct x_partaddr x_partaddr;
struct x_optc1 x_optc1 ;
char   *antwort = "Benutzerverbindungsnachricht, laenger als 32
                   Zeichen" ;
int     retval ;
char    udatap[X_MSG_SIZE] ;
                /* Benutzernachricht mit x_conin erhalten */
    .
    .
    .
/* Ereignisgesteuerte Verarbeitung:
 * x_event() synchron (X_WAIT) wartend
 */
for (;;) {
        switch (event = x_event(&tref, X_WAIT, NULL)) {

        case X_CONIN:
        case X_REPCIN :

            /* Verbindungsaufbauwunsch akzeptieren */

            x_optc1.x_optnr = X_OPTC3 ;
            x_optc1.x_udatap = udatap ;
            x_optc1.x_udatal = sizeof(udatap) ;
            if ((retval=x_conin(&tref, (union x_address
            *)&x_myname,
                (union x_address *)&x_partaddr, &x_optc1))
```

```
            == X_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei x_conin
                tref 0x%x\n",x_error(), tref);
                exit(ERROR);
            }
            if ( retval == X_REPEAT )

              break;
              /* Warten auf X_REPCIN */

            x_optc1.x_udatap = antwort ;
            x_optc1.x_udatal = strlen(antwort) ;
            if (x_conrs(&tref, &x_optc1) == X_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei x_conrs
                tref 0x%x\n",x_error(), tref);
                exit(ERROR);
                }
                .
                .
        case X_DISIN:
            /*
             * Verbindungsabbau durch Partner oder System
             */
            if (x_disin(&tref, &reason, NULL) == X_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei x_disin
                tref 0x%x\n",x_error(), tref);
                exit(ERROR);
             }
             fprintf(stderr, "Verbindungsabbau erhalten, tref
             0x%x,reason %d\n", tref, reason);
                .
                .
        }
}
/*
 * Verbindungsabbau
 */
if (x_disrq(&tref, NULL) == X_ERROR){
    fprintf(stderr, ">>> FEHLER 0x%x bei x_disrq tref 0x%x\n",
            x_error(), tref);
    exit (ERROR);
}
fprintf(stderr,"Verbindung tref 0x%x abgelehnt,bzw. aktiv
            abgebaut.\n", tref);
    .
    .
```

6.4 Verbindungen umlenken

Ankommende Verbindungen für eine lokale TS-Anwendung erhält zunächst der Prozess, der sich als Erster für diese TS-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 TS-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, CMX beim Anmelden mitteilen.

Ablauf beim Umlenken einer Verbindung

Prozess A gibt beim Aufruf *t_redrq()* die Prozess-Nummer 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 Prozess-Nummer von Prozess A mitgeteilt. Will Prozess B die Verbindung nicht haben, kann er sie abbauen oder umlenken, z.B zurück an Prozess A.

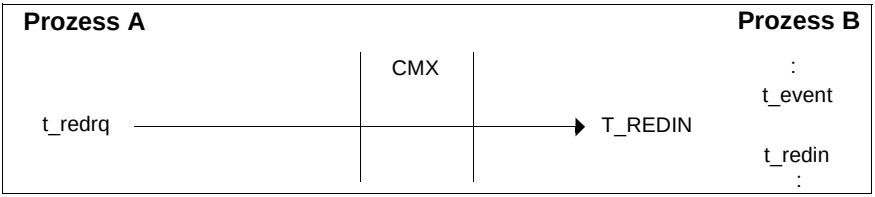

Bild 21: Verbindung umlenken

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

Verbindungsumlenkung an ICMX(NEA)

Soll die TS-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen anstatt *t_redrq()*, *t_redin()* und *t_event()* die Aufrufe *x_redrq()*, *x_redin()* und *x_event()* verwendet werden. Der Prozess B erhält das Ereignis X_REDIN zugestellt.

Für die Übertragung von Benutzerdaten muss in jedem Fall ein Speicherbereich angegeben werden, da der Migrationsservice eine Nachricht zur Übergabe eines internen NEABX-Protokolls aufbauen muss.

6.4.1 Beispiel zur Umlenkung einer Verbindung

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

6.4.1.1 Beispiel zur Umlenkung einer Verbindung an ICMX(L)

```
#include        <stdio.h>
#include        <cmx.h>
#include        <tnsx.h>
  .
#define ERROR   1
  .
int     tref;     /* Transportreferenz */
int     cpid;  /* ID des Prozesses, der Verbdg. erhalten soll */
int     rpid;  /* ID des Prozesses, der Verbdg. abgeben will */
  .
  .

/* Verbindung aktiv umlenken */

if (t_redrq(&tref, &cpid, NULL) == T_ERROR) {
    fprintf(stderr, ">>> FEHLER Ox%x bei t_redrq tref Ox%x\n",
            t_error(), tref);
    exit(ERROR);
}
fprintf(stderr, "Verbindung an #%d umgelenkt\n", cpid);
 .
 .
/* Verbindungsumlenkung entgegennehmen */

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

        case T_REDIN:
             if (t_redin(&tref, &rpid, NULL) == T_ERROR) {
                 fprintf(stderr, ">>> FEHLER Ox%x bei t_redin
                 tref Ox%x\n",t_error(), tref);
                 exit(ERROR);
             }
             fprintf(stderr, "Verbindung von #%d erhalten.\n",
             rpid);
             .
             .
         }
}
```

6.4.1.2 Beispiel zur Umlenkung einer Verbindung an ICMX(NEA)

```
#include        <stdio.h>
#include        <cmx.h>
#include        <tnsx.h>
#include        <neabx.h>
  .
#define ERROR   1
  .
int     tref;   /* Transportreferenz */
int     cpid;   /*ID des Prozesses, der Verbdg. erhalten soll */
int     rpid;   /*ID des Prozesses, der Verbdg. abgeben will */
struct x_optc2 x_optc2 ;
char nachricht[X_RED_SIZE] ;
  .
/* Verbindung aktiv umlenken */
strcpy(nachricht,"privat");
x_optc2.x_optnr = X_OPTC2 ;
x_optc2.x_udatap = nachricht ;
x_optc2.x_udatal = strlen(nachricht) + X_RED_PL ;
if (x_redrq(&tref, &cpid, &x_optc2) == X_ERROR) {
    fprintf(stderr, ">>> FEHLER 0x%x bei x_redrq tref 0x%x\n",
            x_error(), tref);
    exit(ERROR);
}
fprintf(stderr, "Verbindung an #%d umgelenkt\n", cpid);
 .
/* Verbindungsumlenkung entgegennehmen */
for (;;) {
        switch (event = x_event(&tref, X_WAIT, NULL)) {

        case X_REDIN:
            x_optc2.x_optnr = X_OPTC2 ;
            x_optc2.x_udatap = nachricht ;
            x_optc2.x_udatal = sizeof(nachricht) ;
             if (x_redin(&tref, &rpid, &x_optc2) == X_ERROR) {
                 fprintf(stderr, ">>> FEHLER 0x%x bei x_redin
                 tref 0x%x,x_error(), tref);
                 exit(ERROR);
             }
             fprintf(stderr, "Verbindung von #%d erhalten.\n",
             rpid);
             .
             .
         }
}
```

7 Daten übertragen

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

Die Gesamtmenge von Daten, die aus der Sicht der TS-Anwendungen eine logische Einheit bilden, wird als Nachricht bezeichnet oder als TSDU (Transport Service Data Unit). Die Länge einer TSDU ist beliebig (siehe jedoch Abschnitt „Transportsystem-spezifische Besonderheiten“ auf Seite 104).

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

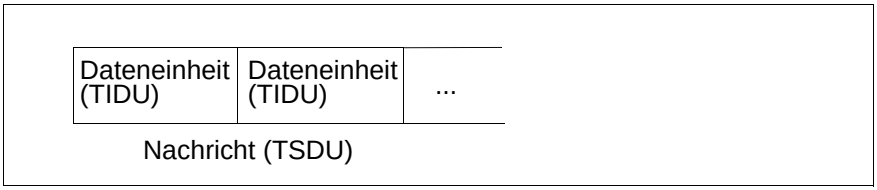

Bild 22: TIDU und TSDU

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

Wenn das Transportsystem diese Option bietet und beide TS-Anwendungen dies beim Verbindungsaufbau vereinbart haben, können sie auch Vorrangdaten austauschen. Vorrangdaten sind kleine Datenmengen, die mit „Vorrang“ vor den Normaldaten transportiert werden. D.h. Vorrangdaten treffen nie später ein als danach gesendete Normaldaten.

Vorrangdaten können immer nur auf einmal übertragen werden. Die Einheit der Vorrangdaten heißt ETSDU (Expedited Transport Service Data Unit).

7.1 Senden und Empfangen von Normaldaten

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

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

Im einfachsten Fall läuft die Datenübertragung so ab:

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

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

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

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

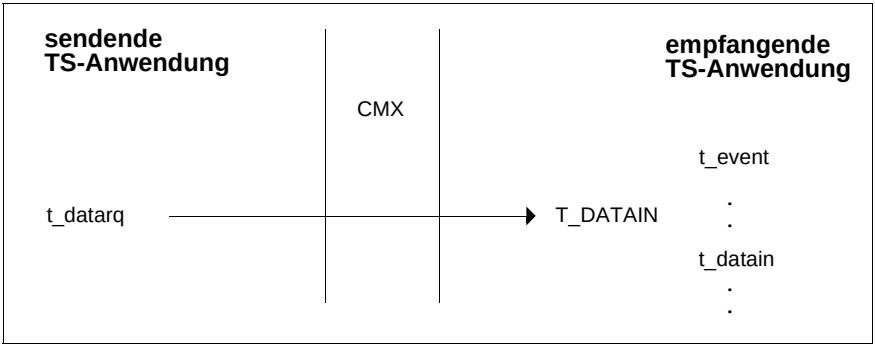

Bild 23: Gewöhnliche Daten übertragen

TSDU-Länge > TIDU-Länge

Wenn eine TSDU größer ist als eine TIDU, muss sie in TIDUs zerlegt werden. Das geht folgendermaßen:

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

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

Jede TIDU kündigt CMX mit einem Ereignis T_DATAIN an. Die Länge einer TIDU kann bei beiden TS-Anwendungen unterschiedlich sein. Deshalb kann es sein, dass die empfangende TS-Anwendung weniger oft *t_datain()* bzw. *t_vdatain()* aufrufen muss als die sendende TS-Anwendung *t_datarq()* bzw. *t_vdatarq()* (oder umgekehrt). Denn die empfangende TS-Anwendung liest TIDUs in „ihrer“ Länge. Das sieht dann wie folgt aus:

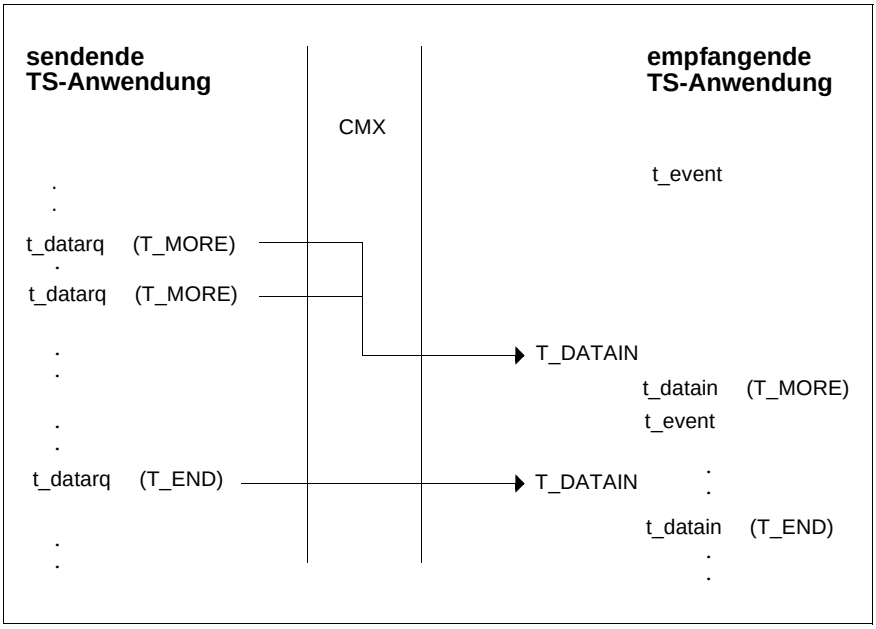

Bild 24: TSDU in mehreren TIDUs

Der Ergebniswert von t_datain() und t_vdatain()

Bei *t_datain()* und *t_vdatain()* muss eine Länge angeben werden, in der die angekommenen Daten gelesen werden sollen. Wenn die angegebene Länge kleiner ist als die TIDU beim Empfänger, gibt der Ergebniswert von *t_datain()* und *t_vdatain()* die Restlänge der Daten in der anstehenden TIDU an. Ist eine TIDU noch nicht vollständig gelesen, muss erneut *t_datain()* bzw. *t_vdatain()* aufgerufen werden, und zwar so oft, bis die TIDU vollständig gelesen ist. Währenddessen darf auch nicht *t_event()* aufgerufen, die Verbindung umgelenkt oder der Datenfluss geregelt werden.

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

Datenübertragung über ICMX(NEA)

Soll die TS-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen anstatt der Aufrufe *t_datarq()*, *t_datain()* und *t_event()* die Aufrufe *x_datarq()*, *x_datain()* und *x_event()* benutzt werden. Entsprechende Aufrufe zu *t_vdatarq() t_vdatain()* gibt es bei ICMX(NEA) nicht. Als Ereignis wird X_DATAIN angezeigt, die Parameterwerte für die Nachrichtenlänge heißen X_MORE und X_END. Zu beachten ist, dass es bei ICMX(NEA) Einschränkungen bezüglich der Datenlängen gibt, insbesondere ist ein stückweises Lesen der Dateneinheit nicht möglich.

Darüber hinaus können die TS-Anwendungen beim Verbindungsaufbau aushandeln, dass bei der Datenübertragung das NEABX-Protokoll in Form von Benutzerdaten ausgetauscht wird. Zur Steuerung bzw. Interpretation des NEABX-Protokolles wird eine Optionsstruktur verwendet.

7.2 Beispiele zur Übertragung von Normaldaten

Die folgenden Programmfragmente zeigen den Programmablauf bei der Übertragung von Normaldaten über ICMX(L) und ICMX(NEA).

7.2.1 Normaldatenübertragung über ICMX(L)

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

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

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

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

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

        case T_DATAIN:
          if ((rc = t_datain(&tref,e_bufpt,&e_bufl,&chain)) ==
             T_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei t_datain
```

```
                tref 0x%x\n",t_error(), tref);
              exit (ERROR);
          }
          :
      }
}
/* Daten senden; s_bufl ist höchstens TIDU-Länge */

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

7.2.2 Normaldatenübertragung über ICMX(NEA)

Das folgende Programmfragment zeigt den Programmablauf bei der Übertragung von Normaldaten über ICMX(NEA). Es wird angenommen, dass die TS-Anwendung beim Aufruf *x_conrq()* bzw. beim Aufruf *x_conrs()* die Verwendung des NEABX-Protokolles in der Datenphase ausgehandelt hat.

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

struct  x_optd1 e_optd1 ;       /* Empfangs Struktur */
char    e_bufpt[8000];          /* Empfangspuffer */
int     e_bufl;                 /* Übertragungslänge */
struct  x_optd1 s_optd1 ;       /* Sende Struktur */
char    s_bufpt[8000];          /* Sendepuffer */
int     s_bufl;                 /* Übertragungslänge */

int     chain;                  /* TSDU-Indikator für
                                   x_datarq(), x_datain() */
int     tref;                   /* Transportreferenz */
 .
 .
```

```
/* Ereignisgesteuerte Verarbeitung:
 * x_event() synchron (X_WAIT) wartend */

for (;;) {
        switch (event = x_event(&tref, X_WAIT, NULL)) {
        .
        .
        /* Daten empfangen; e_bufl ist die TIDU-Länge
        (x_info()) */

        case X_DATAIN:
            e_optd1.x_optnr = X_OPTD2 ;
            e_bufl = sizeof (e_bufpt);
            if((rc = x_datain(&tref,e_bufpt,&e_bufl,&chain,
            (x_optd *)&e_optd1))== T_ERROR) {
                fprintf(stderr, ">>> FEHLER 0x%x bei x_datain
                tref 0x%x\n",x_error(), tref);
                exit (ERROR);
             }
             .
             .
        }
}

/* Daten senden; s_bufl ist höchstens TIDU-Länge */

s_optd1.x_optnr = X_OPTD2 ; /* Angegebene Länge in s_bufl ist
                               Netto */
s_optd1.x_code = X_ASCII;
s_optd1.x_strukt = X_ETXEOT;
if ((rc = x_datarq(&tref, s_bufpt, &s_bufl, &chain, (x_optd
*)&s_optd1))== X_ERROR) {
        fprintf(stderr, ">>> FEHLER 0x%x bei x_datarq tref
                 0x%x\n",x_error(), tref);
        exit(ERROR);
}
```

7.3 Senden und Empfangen von Vorrangdaten

Wenn beim Verbindungsaufbau (siehe Abschnitt „Verbindung aufbauen“ auf Seite 49) der Austausch von Vorrangdaten vereinbart wurde, können die TS-Anwendungen diesen wie folgt vornehmen.

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

– Die sendende TS-Anwendung schickt mit einem Aufruf Vorrangdaten.

– Die empfangende TS-Anwendung erhält das Ereignis T_XDATIN angezeigt. Das ist die Mitteilung, dass Vorrangdaten angekommen sind.

– Die empfangende TS-Anwendung muss die Daten mit dem Aufruf *t_xdatin()* annehmen.

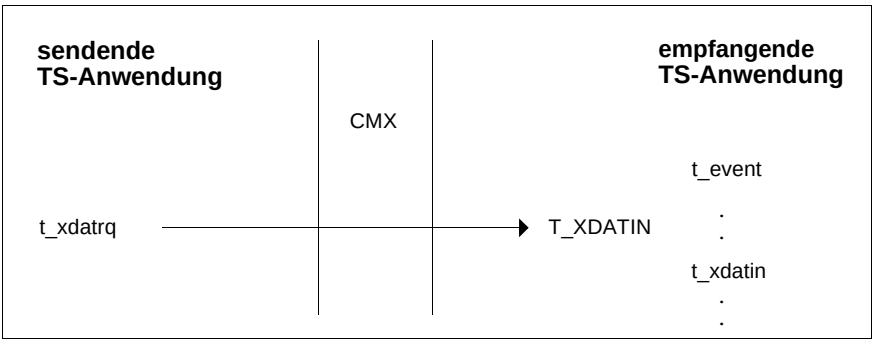

Bild 25: Vorrangdaten übertragen

Der Ergebniswert von t_xdatin()

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

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

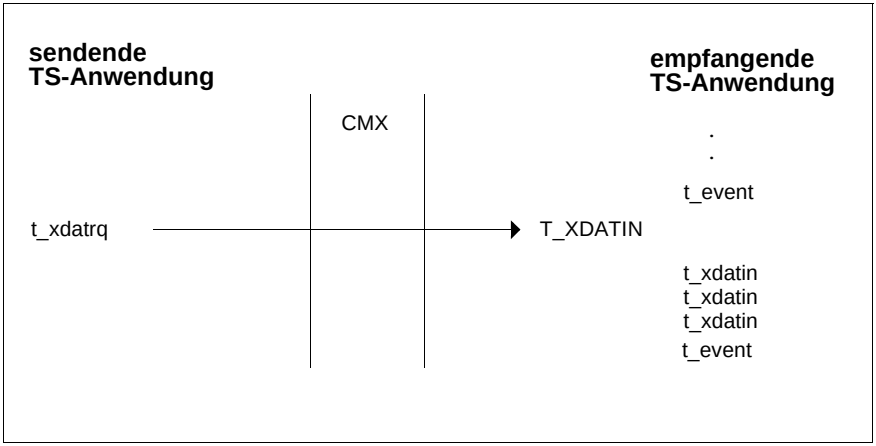

Bild 26: Vorrangdaten stückweise lesen

Vorrangdatenaustausch über ICMX(NEA)

Soll die TS-Anwendung über die Migrationsschnittstelle ICMX(NEA) kommunizieren, so müssen anstatt der Aufrufe *t_xdatrq()*, *t_xdatin()* und *t_event()* die Aufrufe *x_xdatrq()*, *x_xdatin()* und *x_event()* benutzt werden. Als Ereignis wird X_XDATIN gemeldet. Zu beachten ist, dass es bei ICMX(NEA) Einschränkungen bezüglich der Datenlängen gibt. Insbesondere ist ein stückweises Lesen der Vorrangdaten nicht möglich.

Darüber hinaus können die TS-Anwendungen beim Verbindungsaufbau aushandeln, dass bei der Datenübertragung das NEABX-Protokoll ausgetauscht wird. Zur Steuerung bzw. Interpretation des NEABX-Protokolles wird eine Optionsstruktur verwendet. Die entsprechenden Festlegungen sind in Abschnitt „NEABV-Protokoll“ auf Seite 239 beschrieben.

7.4 Fluss-Regelung von Daten und Vorrangdaten

Wenn die TS-Anwendung nicht bereit ist, auf einer Verbindung Daten zu empfangen, dann teilt 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 die TS-Anwendung wieder bereit ist, auf der Verbindung Daten zu empfangen, ruft sie *t_datago()* auf. Der Kommunikationspartner erhält das Ereignis T_DATAGO zugestellt und die TS-Anwendung kann wieder Daten von ihm empfangen. Sie erhält das Ereignis T_DATAIN wieder zugestellt.

In gleicher Weise erfolgt die Fluss-Regelung für Vorrangdaten. Man verwendet dafür die Aufrufe *t_xdatstop()* und *t_xdatgo()*. Die entsprechenden Ereignisse dazu sind T_XDATIN und T_XDATGO.

Zu beachten ist aber:

Wenn man den Vorrangdatenfluss stoppt (mit *t_xdatstop()*), stoppt CMX implizit auch den Fluss für Normaldaten. Wenn man dann den Vorrangdatenfluss wieder freigibt (mit *t_xdatgo()*) bleibt der Fluss für Normaldaten gesperrt! Man muss ihn eigens freigeben (mit *t_datago()*).

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

Welchen Vorteil bringt es, wenn man sich T_DATAIN oder T_XDATIN nicht mehr zustellen lässt?

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

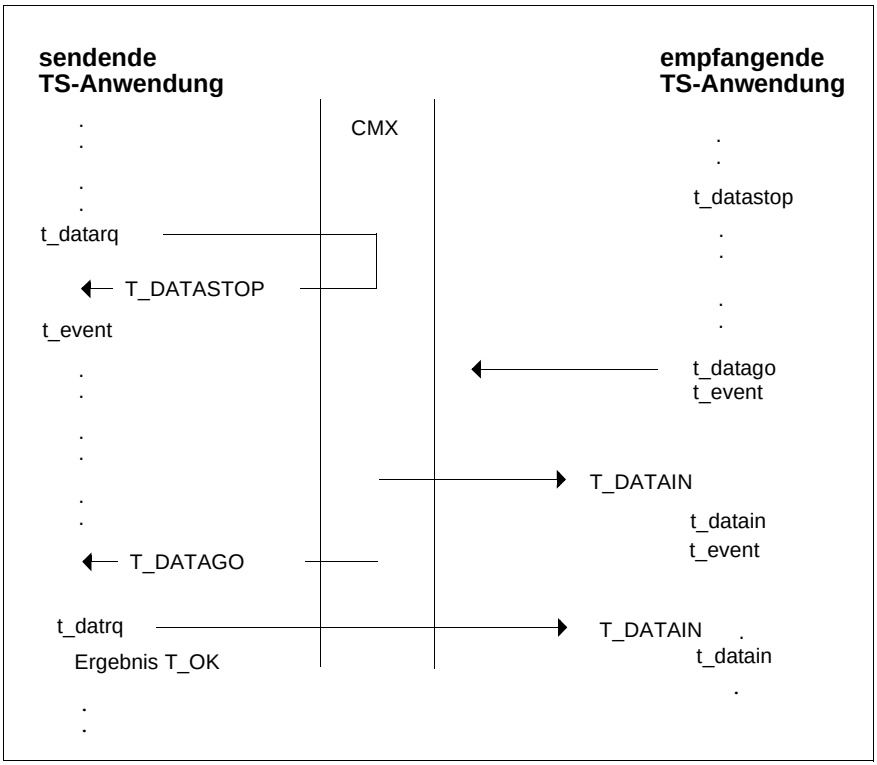

Bild 27: Datenfluss-Regelung beim Sender

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

Datenfluss-Regelung an ICMX(NEA)

Soll die TS-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_ ersetzt ist. Zu beachten ist, dass bei ICMX(NEA) auch beim Aufruf *x_datain()*

bzw. *x_xdatin()* als Ergebnis der Wert X_DATASTOP gemeldet werden kann. Bevor diese TS-Anwendung ihrerseits Daten senden darf, muss sie dazu auf das Ereignis X_DATAGO warten.

8 Programmschnittstelle ICMX(L)

Dieses Kapitel beschreibt die Programmschnittstelle ICMX(L) zum Communications Manager CMX. Es enthält:

– einen Überblick über die Funktionen der Schnittstelle ICMX(L) mit Details zu den Kommunikationsphasen
– Hinweise zur korrekten Verwendung der Funktionen (Zustandsautomaten)
– eine Aufzählung der transportsystem-spezifischen Besonderheiten
– Hinweise zur Verfügbarkeit der Systemoptionen für die Transportsysteme
– die präzise Beschreibung der Funktionsaufrufe von ICMX(L) mit allen Parametern in alphabetischer Reihenfolge.

8.1 Übersicht der Programmschnittstelle

Transport Service ISO 8072

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

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

Tabelle 5: Dienstprimitive in ICMX (L)

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

T-ATTACH
: Anmelden einer TS-Anwendung bei CMX

T-DETACH
: Abmelden einer TS-Anwendung bei CMX

T-ERROR
: Fehlerabfrage

T-REDIRECT
: Umlenken einer Verbindung an einen anderen Prozess

T-FLOWCONTROL
: Fluss-Regelung bei Normaldaten

T-EXPEDITED-FLOWCONTROL
: Fluss-Regelung bei Vorrangdaten

T-EVENT
: TS-Ereignisabfrage

T-INFO
: Information

T-GETADDR
: TRANSPORTADRESSE abfragen

T-GETLOC
: LOKALEN NAMEN abfragen

T-GETNAME
: GLOBALEN NAMEN abfragen

T-CALLBACK
: Rückrufroutine registrieren

T-SETOPT
: Optionen setzen

Der TS erlaubt es zwei TS-Anwendungen, Nachrichten über eine Transportverbindung (TV) auszutauschen. Die verbindungsorientierte Kommunikation bietet den verlust- und duplikatfreien Austausch von Nachrichten unter Beibehaltung der Nachrichtenreihenfolge. Ferner ermöglicht der verbindungsorientierte TS über Verbindungsidentifikationen den Verzicht auf Adress-Übertragung und -verarbeitung in der Datenphase. Eine aufgebaute TV ist (in beiden Endsystemen) durch je eine Transportreferenz (tref) zwischen CMX und TS-Anwendung

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

Jeder TV werden abhängig vom verwendeten TS eine oder mehrere Gerätedateien zugeordnet. Sie werden für die TS-Anwendung nur dadurch sichtbar, dass sie entsprechend viele der verfügbaren Dateikennzahlen (file descriptors) konsumieren. Diese Gerätedateien vereinfachen in CMX die Aufräummaßnahmen bei vorzeitiger Beendigung der TS-Anwendung.

Namen und Adressen

Jede TS-Anwendung hat einen GLOBALEN NAMEN. Unter diesem Namen ist die TS-Anwendung im Netz eindeutig identifizierbar. Die Vergabe der GLOBALEN NAMEN erfolgt per TNS-Administration. Sie muss sicherstellen, dass die Namen aller TS-Anwendungen voneinander verschieden sind.

Die TS-Anwendung arbeitet bei der Identifizierung von Partner-Anwendungen ausschließlich mit GLOBALEN NAMEN.

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

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

ICMX(L)-Funktionen zur Abfrage von Informationen aus dem TS-Directory sind:

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

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

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

t_getaddrpart() und **t_setaddrpart()**
analysiert bzw. modifiziert eine TRANSPORTADRESSE.

t_getlocpart() und **t_setlocpart()**
analysiert bzw. modifiziert einen LOKALEN NAMEN.

In <*cmx.h*> sind die Strukturen *t_myname* und *t_partaddr* definiert. In *t_myname* übernimmt (übergibt) die TS-Anwendung ihren LOKALEN NAMEN vom (an den) TNS, in *t_partaddr* die TRANSPORTADRESSE. Die Strukturen sind wie folgt aufgebaut:

```
struct t_myname {
    char t_mnmode;          /* = T_MNMODE */
    char t_mnres;           /* = 0 */
    short t_mnlng;          /* Länge des ausgefüllten Teils
                               der Struktur t_myname */
    char t_mn[T_MNSIZE];    /* Feld für die T-Selektoren des
                               LOKALEN NAMENS */
 }

struct t_partaddr {
    char t_pamode;          /* = T_PAMODE */
    char t_pares;           /* = 0 */
    short t_palng;          /* Länge des ausgefüllten Teils
                               der Struktur t_partaddr */
    char t_pa[T_PASIZE];    /* Feld für die Partneradresse */
 }
```

Die Elemente der Struktur *t_myname* haben die folgende Bedeutung:

t_mnmode = T_MNMODE
gibt an, dass das Feld *t_mn* einen LOKALEN NAMEN enthält.

t_mnres, *t_mn*[T_MNSIZE]
sind für Sie nicht relevant. Der Inhalt dieser Felder wird nur vom TNS übernommen und an CMX weitergereicht.

t_mnlng
gibt die Länge aller in der Struktur *t_myname* übergebenen Daten an.

Die Elemente der Struktur *t_partaddr* haben die folgende Bedeutung:

t_pamode = T_PAMODE
gibt an, dass das Feld *t_pa* eine TRANSPORTADRESSE enthält.

t_pares, *t_pa*[T_PASIZE]
sind für Sie nicht relevant. Der Inhalt dieser Felder wird nur vom TNS übernommen und an CMX weitergereicht.

t_palng
gibt die Länge aller in der Struktur *t_partaddr* übergebenen Daten an.

LOKALER NAME und TRANSPORTADRESSE können in der Union *t_address* an CMX übergeben oder von CMX übernommen werden.

```
union t_address {
    struct t_myname tmyname;
    struct t_partaddr tpartaddr;
}
```

Fehlerbehandlung und -diagnose

Alle Funktionsaufrufe enden mit einer Rückmeldung. Diese zeigt entweder mit *T_OK* den erfolgreichen Abschluss an, oder informiert durch *T_ERROR* pauschal über einen aufgetretenen Fehler. Die Fehlerabfragefunktion *t_error()*, sofort nach Auftreten eines Fehlers aufgerufen, liefert detailliertere Diagnoseinformation. Alle Fehlermeldungen, die von CMX 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. Den Abbaugrund erhält die TS-Anwendung beim Aufruf von *t_disin()*.

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

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

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

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

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

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

Das Fehlerdecodierprogramm *cmxdec* (siehe „CMX, Betrieb und Administration“ [1] oder [2]) stellt die Klartextaufbereitung über die Kommandozeile zur Verfügung.

Zur Diagnose steht an ICMX(L) ein Verfolger zur Verfügung. Er ist über die Umgebungsvariable CMXTRACE flexibel einstellbar. Der Verfolger protokolliert die Aufrufe mit ihren Argumenten kompakt in temporäre Dateien. Das Aufbereitungsprogramm *cmxl* setzt dann entkoppelt das Protokoll in Klartext um (siehe CMX „Betrieb und Administration“ [1] oder [2]).

TS-Anwendungen, Transportverbindungen und 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 Prozess-Konzept des Systems bleibt dem Implementierer überlassen.

Eine TS-Anwendung kann sich in einem oder mehreren (nicht notwendig verwandten) Prozessen organisieren. Die Prozesse können prinzipiell unabhängig voneinander Transportverbindungen zu fernen TS-Anwendungen unterhalten.

Die Prozesse einer TS-Anwendung können ihre Transportverbindungen untereinander austauschen. Die Transportreferenz einer Transportverbindung 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 Transportverbindung 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 ihrem Wesen nach asynchron: verschiedenste TS-Ereignisse können unabhängig vom Verhalten einer TS-Anwendung auftreten. Zum Beispiel kann eine TS-Anwendung auf einer Transportverbindung Daten senden, während asynchron für eine andere Transportverbindung 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 als TS-Ereignis 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. Der Wartezustand kann durch Angabe einer Wartezeit zeitlich begrenzt werden oder durch Signale wie SIGALRM vorzeitig abgebrochen werden. Der synchrone Mechanismus ist nützlich für TS-Anwendungen, die gleichzeitig mehrere Transportverbindungen unterhalten, damit sie diese nicht pollen müssen.

Bei asynchroner Ausführung kann der Prozess zu ihm genehmen Zeitpunkten nachfragen, etwa am Ende eines Verarbeitungsschrittes, 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, dem Ablaufen der Wartezeit oder dem Eintreffen eines Signals. 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. *t_event()* kehrt mit T_NOEVENT oder T_ERROR zurück. Bei Ablauf der Wartezeit setzt sich der Prozess mit dem TS-Ereignis T_NOEVENT fort. Bei T_CHECK kehrt diese Funktion immer sofort zurück und liefert entweder den Code des gefundenen TS-Ereignisses, oder T_NOEVENT oder T_ERROR.

Folgende asynchrone TS-Ereignisse sind in CMX definiert:

T_NOEVENT
: im asynchronen Fall: kein TS-Ereignis vorhanden.

  im synchronen Fall: Abbruch durch Signal oder Ablauf der Wartezeit.

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

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

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

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 von Normaldaten von einer fernen TS-Anwendung;

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

T_DATAGO
: Aufheben einer durch die Fluss-Regelung gesetzten Sendesperre für Normaldaten und Vorrangdaten;

T_XDATGO
: Aufheben einer durch die Fluss-Regelung 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.

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

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

Tabelle 6: TS-Ereignisse und Funktionen

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.

Signalisierung bei asynchroner Ereignisverarbeitung

Für die asynchrone Ereignisverarbeitung stellt CMX optional einen Signalmechanismus zur Verfügung, durch den unnötige *t_event()*-Aufrufe mit Ergebnis T_NOEVENT vermieden werden können. Ist die Signalisierung eingeschaltet, so wird dem Prozess jedes Eintreffen eines Ereignisses durch ein Signal angezeigt. Die Signalisierung erfolgt asynchron zum Prozess-Verlauf. Nach Eintreffen eines Signals wird entweder eine von Ihnen definierte Signalroutine oder eine CMX-eigene Signalroutine durchlaufen. Die CMX-eigene Signalroutine bewirkt die Protokollierung der Signale im CMX-Verfolger. Nach Ablauf der Signalroutine sollte der Prozess mit *t_event()* prüfen, ob ein Ereignis ansteht und dieses geeignet verarbeiten.

Per Voreinstellung ist die Signalisierung ausgeschaltet. Aktivieren und Steuern kann man sie über die Umgebungsvariable CMXINIT.

CMXINIT="-s" aktiviert die Signalisierung mit Signal 22 (SIGIO).

CMXINIT="-S n" aktiviert die Signalisierung mit Signal n (n = Dezimalzahl).

n sollte geeignet gewählt werden. Nicht alle Signale sind mit Signalroutinen abfangbar.

In C-Programmen kann CMXINIT gesetzt werden durch:

```
putenv("CMXINIT='-s'"); bzw. putenv("CMXINIT='-S  n'";)
```

CMXINIT wird innerhalb eines Prozesses einmal vor dem ersten Aufruf an ICMX(L) ausgewertet.

Für multithreaded Anwendungen steht die Signalisierung nicht zur Verfügung. Die entsprechenden Optionen in CMXINIT werden ignoriert.

Anmeldung/Abmeldung

Die Kommunikation eines Prozesses über CMX wird aktiviert, indem sich der Prozess bei CMX anmeldet. Bei der ersten Anmeldung wird für diesen Prozess eine Gerätedatei eröffnet. Über diese Gerätedatei werden Aufträge zwischen den CMX-Bibliotheksfunktionen und dem Betriebssystem ausgetauscht. Durch den ersten Prozess, der sich für eine TS-Anwendung anmeldet, wird diese TS-Anwendung erzeugt. Dabei wird ein Dienstzugriffspunkt (Transport Service Access Point TSAP) eingerichtet, an dem der TS zur Verfügung steht. Mit der Anmeldung des ersten Prozesses wird die TS-Anwendung an diesen TSAP gebunden. Dem TSAP wird der LOKALE NAME der TS-Anwendung zugeordnet. Sie wird damit aus dem Netz adressierbar. Bei der Abmeldung werden noch bestehende TVen und der TSAP abgebaut, die Prozess-Umgebung aufgelöst und belegte Betriebsmittel für zukünftige Verwendung freigegeben.

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

Verbindungsaufbau, -abbau und -umlenkung

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

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

Jede der TS-Anwendungen sowie CMX können jederzeit die TV abbauen. Dies wird von den TS-Anwendungen nicht verhandelt, sondern von CMX sofort vollzogen. Der anderen TS-Anwendung (oder beiden, wenn CMX die TV abbaut) wird eine Verbindungsabbauanzeige zugestellt, die weder beantwortet noch

abgewehrt werden kann. Alle Fehlerfälle in den Transportsystemen werden von CMX durch einen Abbau der betroffenen TVen angezeigt. CMX garantiert nicht, dass Daten, die zum Zeitpunkt der Anforderung des Abbaus noch unterwegs sind, noch zugestellt werden.

Die Verbindungsumlenkung ist ein lokaler Dienst in CMX, 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 „Zustände von TS-Anwendungen und erlaubte Übergänge“ auf Seite 98) 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 ist.

Die entsprechenden Funktionen sind im Folgenden beschrieben.

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

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

t_redrq()
: lenkt die TV an einen Prozess derselben TS-Anwendung um, für den abgebenden Prozess ist sie dann 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.

Datenaustausch und Fluss-Regelung

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

– Normaldaten und (falls vereinbart) Vorrangdaten senden oder

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

Der Datentransfer findet 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 Bytes beschränkt, sie werden mit Vorrang zum Strom der Normaldaten übertragen und in eigenen Warteschlangen geführt. CMX garantiert nur, dass Vorrangdaten nie später bei der empfangenden TS-Anwendung eintreffen als die Normaldaten, die nach ihnen abgesendet wurden. Pro Aufruf kann an CMX höchstens eine komplette ETSDU übergeben werden.

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

den sein und hängt von der TV ab. 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 aus einer TSDU (die Ankunft einer ETSDU) über das TS-Ereignis T_DATAIN (T_XDATIN) angezeigt. Sie holt dann die TIDU (ETSDU) mit einem entsprechenden Funktionsaufruf ganz oder teilweise ab. Gegebenenfalls kann oder muss sie mehrere derartige Aufrufe absetzen, um eine TIDU (ETSDU) von CMX zu übernehmen.

Die Übertragung von TIDUs (ETSDUs) unterliegt Fluss-Regelungsmechanismen, 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) zwar verarbeitet ist, der Fluss der TIDUs (ETSDUs) aber gesperrt wurde. Es kann keine TIDU (ETSDU) mehr gesendet werden, bis der Fluss wieder freigegeben wird. Dies wird durch das TS-Ereignis T_DATAGO (T_XDATGO) angezeigt.

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

Die folgenden Funktionen realisieren Datenaustausch und (aktive) Fluss-Regelung:

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

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

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

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

t_xdatrq()
fordert die Übertragung einer (eventuell teilgefüllten) 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_datastop()
sperrt empfangsseitig den Fluss der Normaldaten auf einer Verbindung; das TS-Ereignis T_DATAIN wird von CMX für diese Verbindung nicht mehr angezeigt;

t_datago()
gibt empfangsseitig den (gesperrten) Fluss der Normaldaten und Vorrangdaten auf einer Verbindung wieder frei; die TS-Ereignisse T_DATAIN und T_XDATIN werden von CMX 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 TS-Ereignisse T_XDATIN und T_DATAIN nicht mehr an;

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

Informationsdienst

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

t_info()
liefert für eine eingerichtete TV die Länge der TIDU. Die TIDU steht in der Regel erst dann fest, wenn der Verbindungsaufbau vollständig abgeschlossen worden ist.

Zentraler Wartepunkt

TS-Anwendungen erwarten oft neben den CMX-Ereignissen anwendungsspezifische Ereignisse. In der Rückrufroutine ist es möglich gleichzeitig auf CMX-Ereignisse und Ereignisse der Anwendung zu warten. Somit kann ein zentraler Wartepunkt definiert werden.

t_callback()
: übergibt an CMX einen Zeiger auf eine Routine in der Anwendung, die während der Ausführung des *t_event()* aufgerufen wird.

Optionen Management

Nur in der CMX-Bibliothek können derzeit Optionen gesetzt werden.

t_setopt()
: setzt oder löscht die Trace-Optionen der jeweiligen Anwendung.

8.2 Zustände von TS-Anwendungen und Zustandsübergänge

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

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

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

1. Phase der Kommunikation: Prozess anmelden

   Der Prozess existiert, er ist noch nicht oder nicht mehr bei CMX angemeldet.

2. Phase der Kommunikation: Verbindung aufbauen

   Der Prozess ist bei CMX angemeldet und es besteht keine Verbindung. Eine Verbindung kann aufgebaut werden.

3. Phase der Kommunikation: Datentransfer

   Die Verbindung ist aufgebaut. Der Prozess kann Daten senden und empfangen.

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

– Normaldaten senden

– Normaldaten empfangen

– Vorrangdaten senden

– Vorrangdaten empfangen

Der Prozess befindet sich, wenn er diese Phase erreicht, zu jeder Zeit in jedem Teilbereich in genau einem Zustand. Es sind nur bestimmte Kombinationen von Zuständen dieser Teilbereiche erlaubt. D. h. ein Zustandsübergang innerhalb eines Teilbereichs kann einen Zustandsübergang in einem anderen Teilbereich bedingen. Wie die einzelnen Zustände in den Teilbereichen verknüpft sind,

kann man anhand der Bedingungen für Zustandsübergänge ablesen (siehe unten). Ist der Austausch von Vorrangdaten für die Verbindung nicht vereinbart, so befindet sich der Prozess nur in Zuständen der oberen beiden Teilbereiche.

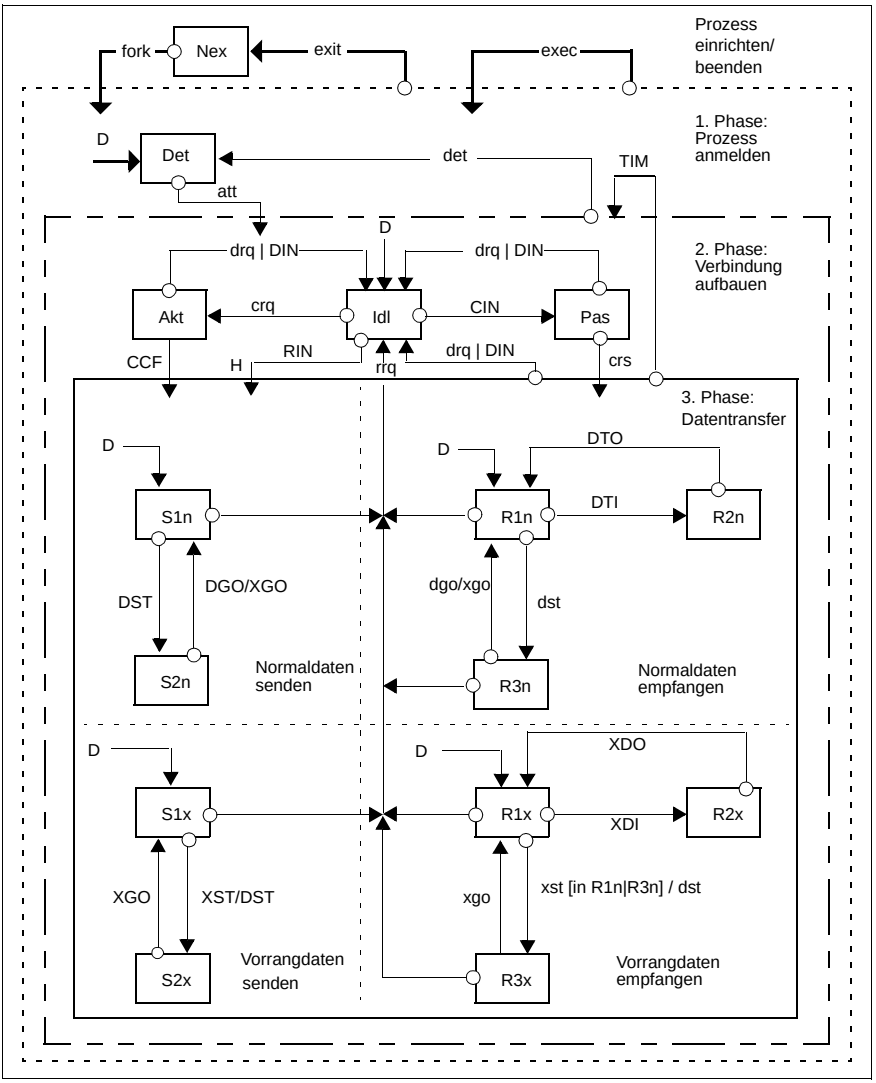

Bild 28: Zustände von TS-Anwendungen und erlaubte Übergänge

Die Pfeile zwischen den Rechtecken zeigen die möglichen Übergänge.
B bezeichnet die Bedingung für den Übergang von Ausgangszustand in den Zielzustand (Ausgangszustand - Zielzustand). Die Übergänge sind nur in der vom Pfeil angegebenen Richtung möglich. Im Folgenden sind zunächst die Abkürzungen erklärt, die im Diagramm verwendet wurden.

Abkürzungen für die Zustände

Nex der Prozess existiert nicht (mehr)

Det die TS-Anwendung ist noch nicht bei CMX angemeldet bzw. die TS-Anwendung ist bei CMX abgemeldet.

Id lGrundzustand zum Verbindungsaufbau und zum Entgegennehmen einer Verbindungsumlenkung bzw. eine vorher bestehende Verbindung wurde abgebaut.

Akt Warten auf das Ereignis T_CONCF nach dem Aufruf *t_conrq()* (aktiver Verbindungsaufbau).

Pas ein Ereignis T_CONIN eingetroffen (passiver Verbindungsaufbau).

S1n Grundzustand für *t_datarq()* bzw. *t_vdatarq()*.

S2n Normaldatenfluss gesperrt

R1n Grundzustand für *t_datain()*

R2n T_DATAIN angezeigt

R3n T_DATAIN gesperrt

S1x Grundzustand für *t_xdatrq()*

S2x Vorrangdatenfluss gesperrt

R1x Grundzustand für *t_xdatin()*

R2x T_XDATIN angezeigt

R3x T_XDATIN gesperrt

Abkürzungen für die Übergangsbedingungen

fork Prozess wird eingerichtet

exec Prozess-Überlagerung

exit Prozess-Beendigung

Folgende Übergänge sind durch Aufruf einer CMX-Funktion bedingt:

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

Folgende Übergänge sind durch die Übernahme von Ereignissen bedingt:

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

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

DST T_DATASTOP bei *t_datarq()* bzw. *t_vdatarq()*

XST T_XDATSTOP bei *t_xdatrq()*

DTO 0 bei *t_datain()* bzw. *t_vdatain()* (aktuelle TIDU vollständig gelesen)

XDO bei *t_xdatin()* (ETSDU vollständig gelesen)

TIM t_timeout (Inaktivzeit der Verbindung abgelaufen)

8.2.1 Erläuterungen zu den möglichen Zustandsübergängen

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

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

Zustandsübergänge dieser Art sind:

– fork
  Wird *fork()* in einem beliebigen Zustand des Prozesses aufgerufen, so wird im Kindprozess der Zustand Det (Prozess noch nicht bei CMX angemeldet) eingenommen. Der Zustand des Vaters bleibt erhalten.

– exec
  Wird *exec()* in einem beliebigen Zustand des Prozesses aufgerufen, so geht der Prozess in den Zustand Det (Prozess abgemeldet) über. Er verliert alle Anmeldungen und Verbindungen.

– exit
  Wird *exit()* aufgerufen, so beendet sich der Prozess. Alle Verbindungen werden von CMX abgebaut.

– det
  Ruft der Prozess in einem beliebigen Zustand *t_detach()* auf, so geht er in den Zustand Det über. CMX baut seine Verbindungen ab.

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

– TIM
  Wurde beim Datentransfer die durch den Parameter *t_timeout* festgelegte Inaktivzeit der Verbindung überschritten, so geht der Prozess in Zustand Idl der 2. Phase über.

Zustandsübergänge innerhalb der 3. Phase (Datentransfer)

Im Folgenden werden die Verknüpfungen von Zustandsübergängen in den Teilbereichen der 3. Phase beschrieben. Der Zustand, den der Prozess im Teilbereich „Normaldaten senden“ einnimmt, ist abhängig von seinem Zustand im Teilbereich „Vorrangdaten senden“ und umgekehrt. Der Zustand, den der Prozess im Teilbereich „Normaldaten empfangen“ einnimmt, ist abhängig von seinem Zustand im Teilbereich „Vorrangdaten empfangen“ und umgekehrt.

Folgende Verknüpfungen bestehen zwischen den Zuständen der vier Teilbereiche:

DGO/XGO (DGO löst XGO aus)
  Das Ereignis T_DATAGO löst T_XDATGO aus. Mit dem Normaldatenfluss wird der Vorrangdatenfluss freigegeben, sofern er gesperrt ist.

  Somit löst der Übergang S2n → S1n den Übergang S2x → S1x aus.

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

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

xst[in R1n|R3n]/dst

Befindet sich der Prozess im Zustand R1x, so kann er nur *t_xdatstop()* aufrufen, wenn er sich im Teilbereich „Normaldaten empfangen“ im Zustand R1n oder R3n befindet. Er löst dabei *t_datastop()* aus. D.h der Vorrangdatenfluss kann von dem Prozess nur gesperrt werden, solange kein T_DATAIN angezeigt wird. Mit dem Vorrangdatenfluss wird implizit der Normaldatenfluss gesperrt (R1x → R3x löst R1n → R3n aus).

8.3 Transportsystem-spezifische Besonderheiten

Dieser Abschnitt beschreibt Besonderheiten zum CMX-API, die an das verwendete Transportsystem gebunden sind.

Setzbare Socketoption bei Verbindungen TCP/IP mit RFC1006

Mit der Umgebungsvariablen CMXSOCKET kann der KEEPALIVE-Mechanismus in TCP für alle in der entsprechenden Prozessumgebung aufzubauenden TCP-Verbindungen aktiviert werden:

CMXSOCKET=-K1; export CMXSOCKET

Falls der KEEPALIVE-Mechanismus für eine TCP-Verbindung aktiviert ist und sobald auf dieser TCP-Verbindung während eines KEEPALIVE-Zeitintervalls keine Daten übertragen worden sind, überprüft TCP mit Testpaketen, ob der Partner noch antwortet. Wenn das Testergebnis negativ ist, baut TCP die Verbindung lokal ab. Die CMX-Anwendung erhält dann eine Verbindungsabbau-Anzeige mit dem Grund T_RLCONNLOST (Verlust der Netzverbindung).

Das KEEPALIVE-Zeitintervall wird durch das Betriebssystem vorgegeben. Es ist in Solaris eine Systemvariable, die auf zwei Stunden voreingestellt ist.

In Solaris ändern Sie den Wert der TCP-Variablen *tcp_keepalive_interval* (in Millisekunden) mit dem Kommando
ndd -set /dev/tcp tcp_keepalive_interval <neuer wert>.

Zeitüberwachung der Antwort auf die Verbindungsaufbau-Anforderung

Bei allen Transportsystemen außer bei der lokalen Kommunikation wird die Antwort auf ein Protokollelement, das die Verbindung anfordert, (z. B. "call request packet" bei X.25, "CR TPDU" bei IS 8073) zeitüberwacht.

Nach Ablauf des protokollbedingten oder TSP-spezifisch konfigurierbaren Timers (Voreinstellung 2 bis 3 Minuten) erhält die rufende Anwendung ein CMX-Event T_DISIN mit Verbindungsabbau-Grund T_RLNORESP. Außerdem wird ein Protokollelement, das den Verbindungsabbau anfordert, an das Partnersystem geschickt.

Bei manchen Transportsystemen wird zusätzlich noch die Reaktion auf eine Verbindungsaufbau-Anzeige (CMX-Event T_CONIN) zeitüberwacht. Wenn die Anwendung nicht innerhalb einer angemessenen Zeit mit *t_conrs()* oder *t_disrq()*

antwortet, erhält sie ein T_DISIN mit Abbaugrund T_RUNKNOWN oder T_USER, abhängig davon, welches Transportsystem verwendet wurde, und ob im lokalen System oder im Partnersystem der kürzere Timer eingestellt war.

Durch das NEA-Protokoll bedingte Besonderheiten

Wenn beim NEA-TSP drei mal Vorrangdaten gesendet werden, die vom Partner nicht abgeholt werden, erfolgt ein Verbindungsabbau.

8.4 Systemoptionen und Nachrichtenlänge

Beim Erstellen von TS-Anwendungsprogrammen sollten Sie beachten, dass die Systemoptionen „Benutzerdatenaustausch beim Verbindungsaufbau und -abbau“ und „Austausch von Vorrangdaten“ nicht von jedem Transportsystem (CCP-Profil) unterstützt werden. Bei den Transportsystemen, die diese Systemoptionen unterstützen, ist die erlaubte Länge der Benutzerdaten bzw. der Vorrangdateneinheit verschieden.

Die Freigabemitteilung enthält genaue Informationen dazu, welche CCP-Profile die Systemoptionen unterstützen und Angaben zur unterstützten Länge für Benutzerdaten, Vorrangdaten und Nachricht.

8.4.1 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. Adressieren Sie die TS-Anwendung ausschließlich mit Hilfe des TNS. 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 Funktion *prototyping* wird von CMX unterstützt. Somit wird das Programm bereits während der Übersetzung auf Korrektheit in der übergebenen Parameterstruktur hingewiesen.

Das Design von ICMX erlaubt einen ereignisgesteuerten Programmablauf. Er ist speziell darauf ausgelegt, die Programmierung einer Ereignisschleife zu ermöglichen, in der auf die einzelnen Ereignisse speziell Rücksicht genommen wird.

Beispiel

Ein ereignisgesteuertes Design für zwei TS-Anwendungen ist in dem folgenden Programmskelett aufgezeigt.

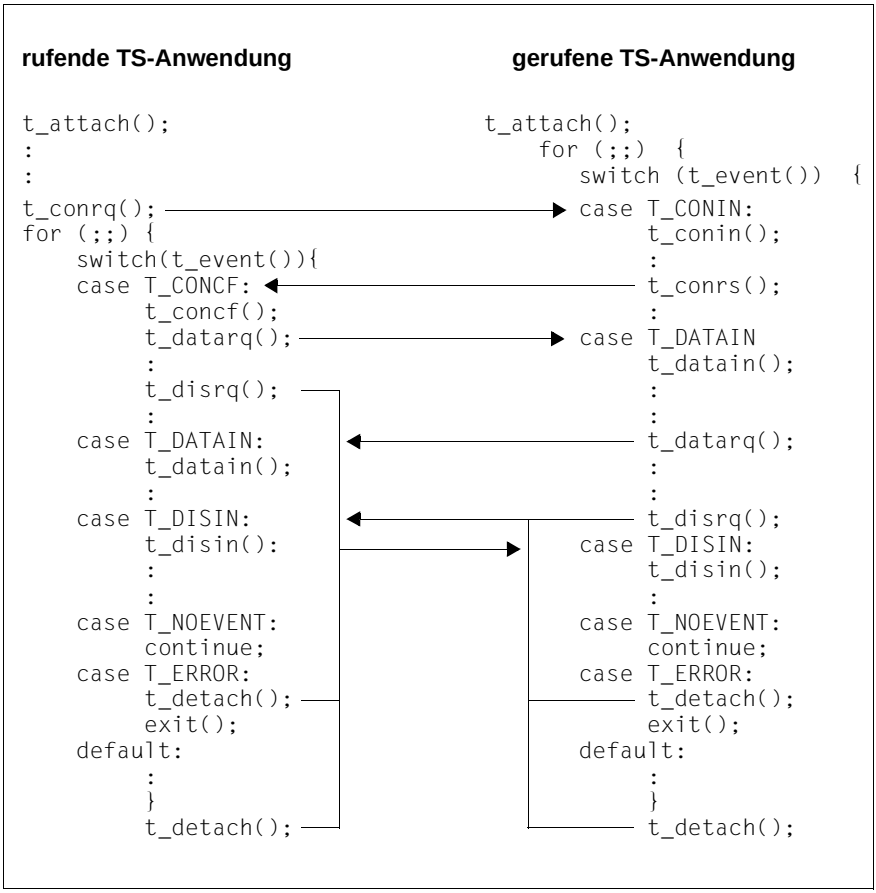

Bild 29: Ereignisgesteuertes Design für zwei TS-Anwendungen

8.4.2 Zusätzliche Funktionalitäten „Betrieb ohne TNS / Erstellung von Schablonen“

Mit den Funktionen *t_getloc()* bzw. *t_getaddr()* können durch Angabe des Wertes NULL im Parameter *globname* und des Wertes OPTG7 im Parameter *t_optnr* innerhalb der Struktur *t_optg7* so genannte Schablonen generiert werden, d. h. LOKALE NAMEN bzw. TRANSPORTADRESSEN mit einem leeren Inhalt. Die Anwendung selbst sorgt dann mit den Funktionen *t_getlocpart()* bzw. *t_getaddrpart()* für die Darstellung dieses LOKALEN NAMENS bzw. dieser TRANSPORTADRESSE in einer Datenstruktur (struct t_addrpart) der Anwendung.

Mit den Funktionen *t_setlocpart()* bzw. *t_ setaddrpart()* kann dann der Inhalt dieser Adresse, der durch die Applikation modifiziert wurde, temporär im Speicher der Applikation geändert werden und anschließend der Funktion *t_attach()* mitgegeben werden im Parameter *name* bzw. der Funktion *t_conrq()* mitgegeben werden in den Parametern *fromaddr* bzw. *toaddr*.

8.4.2.1 Anwendungszenario / Programmskelett

```
struct t_myname MN, newMN;
struct t_partaddr PA, newPA;

MN=t_getloc (NULL;..);  /* Holen einer Schablone eines LOKALEN NAMENS */
        :
t_getlocpart (MN,..);   /* Zerlegen der Schablone
                           in eine Struktur t_addrpart               */

/* Setzen der programmierspezifischen Information
   in die Struktur t_addrpart                                        */

t_setlocpart (MN, newMN..);    /* Generieren des mit Inhalt
                                  gefüllten LOKALEN NAMENS newMN     */

t_attach (MN,..)

PA=t_getaddr (NULL,...);       /* Holen einer Schablone einer
                                  TRANSPORTADRESSE                   */
        :
t_getaddrpart (PA,..);         /* Zerlegen der Schablone in eine
                                  Struktur t_addrpart                */

/* Setzen der programmspezifischen Information
   in die Struktur t_addrpart                                        */

t_setaddrpart (PA, newPA..);   /* Generieren der mit Inhalt
                                  gefüllten TRANSPORTADRESSE newPA   */
t_conrq (newMN, newPA,..);
```

8.5 Konventionen

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

1. Alle Identifier, die mit _ (Unterstrich) beginnen, sind reserviert für die Systemsoftware.
2. Alle Identifier, die mit t_ oder ts oder Ts beginnen sind für CMX reserviert.
3. Alle Präprozessordefinitionen, die mit T_ oder TS_ beginnen, sind für CMX reserviert.
4. Auf Anforderung des Benutzers werden (von CMX-Komponenten im Kernel) Signale verschickt und in der CMX-Bibliothek eingefangen (in der Regel SIGIO und/oder SIGTERM). Die Verwendung von benutzereigenen Signalroutinen sollte daher sorgfältig programmiert werden.

8.6 ICMX(L) - Funktionsaufrufe

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

Folgende Kennzeichnungen werden bei der Parameterbeschreibung verwendet:

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

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

<> kennzeichnet Parameter, in denen der Aufrufer einen Wert bereitstellen muss, der dann von CMX modifiziert wird.

Die Modifikation erfolgt im Allgemeinen nur bei positivem Ablauf. Bei negativem Verlauf bleibt der Wert unverändert.

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

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

8.6.1 t_attach - Anmelden eines Prozesses bei CMX (attach process)

t_attach() meldet den laufenden Prozess bei CMX an. Durch die Parameter, die beim Aufruf von *t_attach()* übergeben werden, legt man fest:

– für welche TS-Anwendung sich der Prozess anmeldet,

– welche Arten des Verbindungsaufbaus (passiv, aktiv, umgelenkte Verbindung entgegennehmen) dem Prozess in dieser TS-Anwendung möglich sind,

– wie viele Verbindungen der Prozess in dieser TS-Anwendung gleichzeitig haben darf.

Die TS-Anwendung, für die sich der Prozess anmeldet, hat einen GLOBALEN NAMEN und einen oder mehrere T-Selektoren, die im lokalen System eindeutig sind. Die T-Selektoren werden zusammengefasst zum LOKALEN NAMEN. Der LOKALE NAME muss CMX beim *t_attach()* als Parameter übergeben werden. Mit Hilfe des Aufrufs *t_getloc()* und des GLOBALEN NAMENS der TS-Anwendung kann der LOKALE NAME vom TNS abgefragt und in einem Datenbereich bereitgestellt werden. Beim *t_attach()* ist dann der Zeiger auf diesen Datenbereich zu übergeben.

Der laufende Prozess kann sich durch wiederholte *t_attach()*-Aufrufe für mehrere verschiedene TS-Anwendungen bei CMX anmelden.

Ebenso können sich mehrere Prozesse für dieselbe TS-Anwendung, d. h. mit demselben LOKALEN NAMEN bei CMX anmelden. Der erste Prozess, der sich für eine TS-Anwendung anmeldet, erzeugt diese TS-Anwendung.

CMX nimmt Verbindungswünsche für eine TS-Anwendung aus dem Netz entgegen, sobald sich ein Prozess in dieser TS-Anwendung für die Annahme von Verbindungsanzeigen bei CMX angemeldet hat, d. h. wenn man T_PASSIVE in *t_apmode* gesetzt hat.

Haben sich mehrere Prozesse für eine TS-Anwendung mit T_PASSIVE angemeldet, so stellt CMX alle Verbindungsanzeigen für diese TS-Anwendung zunächst dem Prozess zu, der sich als erster mit T_PASSIVE für diese TS-Anwendung angemeldet hat. Erst wenn die Anzahl der Verbindungen erreicht ist, die dieser Prozess für diese TS-Anwendung haben darf, werden die eintreffenden Verbindungsanzeigen einem der anderen Prozesse zugestellt. Die Reihenfolge hierfür ist nicht definiert.

Hinweise

Beim ersten *t_attach()* wird im laufenden Prozess eine Dateikennzahl (file descriptor) belegt. Diese bleibt während der Lebensdauer des Prozesses belegt.

T_OK mit mehreren T-Selektoren bedeutet, dass die Anmeldung für mindestens einen T-Selektor erfolgreich war.

```
#include <cmx.h>
int   t_attach (const struct t_myname *name,
                t_opta *opt);
```

-> name

Zeiger zu einer Struktur *t_myname* mit dem LOKALEN NAMEN der TS-Anwendung. Den LOKALEN NAMEN liefert der TNS als Eigenschaft zum GLOBALEN NAMEN der TS-Anwendung.

<> opt

Für den Parameter *opt* ist der Wert NULL oder der Zeiger zu einer Union mit Benutzeroptionen anzugeben.

Wird *opt* = NULL angegeben, so setzt CMX die angegebenen Standardwerte.

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

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

      struct t_opta2 {
->       int t_optnr;    /* Options-Nr. */
->       int t_apmode;   /* Prozess-Mode */
->       int t_conlim;   /* Verbindungsanzahl */
->       int t_uattid;   /* Benutzerreferenz der
                            Anmeldung */
<-       int t_attid;    /* CMX-Referenz der Anmeldung */
<-       int t_ccbits;   /* Bitliste betroffener CCs' */
<-       int t_sptypes;  /* Betroffene Adress-Formate */
      }
```

```
      struct t_opta5 {
->       int t_optnr;               /* Options-Nr. */
->       int t_apmode;              /* Prozess-Mode */
->       int t_conlim;              /* Verbindungsanzahl */
->       int t_uattid;              /* Benutzerreferenz der
                                       Anmeldung */
<-       int t_attid;               /* CMX-Referenz der Anmeldung */
<-       int t_ccbits;              /* Bitliste betroffener CCs' */
<-       int t_sptypes;             /* Betroffene Adress-Formate */
<-       int t_evref;               /* Referenz Punkt */
      }

      struct t_opta6 {
->       int t_optnr;               /* Options-Nr. */
->       int t_apmode;              /* Prozess-Mode */
->       int t_conlim;              /* Verbindungsanzahl */
->       int t_uattid;              /* Benutzerreferenz der
                                       Anmeldung */
<-       int t_attid;               /* CMX-Referenz der Anmeldung */
<-       struct t_cclst *t_cclist;  /* Adresse der CC-Liste */
<-       int t_sptypes;             /* Betroffene Adress-Formate */
      }

      struct t_opta7 {
->       int t_optnr;                /* Options-Nr. */
->       int t_apmode;               /* Prozess-Mode */
->       int t_conlim;               /* Verbindungsanzahl */
->       int t_uattid;               /* Benutzerreferenz der Anmeldung */
<-       int t_attid;                /* CMX-Referenz der Anmeldung */
<-       struct t_cclst *t_cclist;   /* Adresse der CC-Liste */
<-       int t_sptypes;              /* Betroffene Adress-Formate */
<-       int t_evref;                /* Referenz-Punkt */
->       char *t_hostname;           /* Hostname, der der IP-Adresse
                                        eines lokalen IP-Interfaces
                                        entspricht, */
      }
```

t_optnr

Optionsnummer. Anzugeben ist:

T_OPTA1 bei t_opta1

T_OPTA2 bei t_opta2

T_OPTA5 bei t_opta5

T_OPTA6 bei t_opta6

T_OPTA7 bei t_opta7

t_apmode

t_apmode legt fest, welche Arten des Verbindungsaufbaus dem Prozess in dieser TS-Anwendung möglich sind.

Zulässige Werte sind:

T_ACTIVE
Der Prozess soll aktiv Verbindungen aufbauen.

T_PASSIVE
Der Prozess soll passiv auf Verbindungsaufbauwünsche warten.

T_REDIRECT
Der Prozess soll umgelenkte Verbindungen annehmen.

Diese Werte können durch bitweises ODER (|) kombiniert werden, z. B. T_ACTIVE | T_PASSIVE.

Standardwert bei Angabe von *opt* = NULL:
T_ACTIVE | T_PASSIVE | T_REDIRECT

t_conlim

Für *t_conlim* ist die maximale Anzahl an Verbindungen anzugeben, die der Prozess in dieser TS-Anwendung gleichzeitig haben darf.

Wird *t_conlim* = T_NOLIMIT angegeben, so kann der Prozess das installationsspezifische Maximum an Verbindungen gleichzeitig halten.

Standardwert bei Angabe von *opt* = NULL: 1

t_uattid

Im Feld *t_uattid* kann man CMX eine beliebige Benutzerreferenz dieser Anmeldung übergeben. Sie wird im Folgenden als Option beim *t_event* von CMX zurückgeliefert, d. h. wenn der laufende Prozess das Eintreffen eines Ereignisses von CMX abfragt.

Mit Hilfe dieser Benutzerreferenz kann ein Prozess, der mehrere TS-Anwendung steuert, die eintreffenden Ereignisse einfacher den entsprechenden Anmeldungen zuordnen.

Standardwert bei Angabe von *opt* = NULL: 0

t_attid

Dieses Feld dient Verfolger- und Diagnosezwecken. Es wird ausschließlich zur Protokollierung verwendet.

Im Feld *t_attid* liefert CMX die CMX-interne Referenz der Anmeldung zurück.

t_ccbits
: Dieses Feld dient Verfolger- und Diagnosezwecken. Es wird ausschließlich zur Protokollierung verwendet.

  Die Bedeutung ist *<cmx.h>* zu entnehmen.

t_sptypes
: Dieses Feld dient Verfolger- und Diagnosezwecken. Es wird ausschließlich zur Protokollierung verwendet.

  In *t_sptypes* liefert CMX eine bitverschlüsselte Liste der Adress-Formate, für die diese Anmeldung erfolgreich war.

  Die Bedeutung ist *<cmx.h>* zu entnehmen.

t_evref
: Ereignisreferenzpunkt. Wird nur aus Kompatibilität zu CMX in BS2000/OSD zugelassen und in CMX in UNIX (Solaris und Reliant UNIX) nicht unterstützt.

t_cclist
: Dieser Pointer auf eine CC-Liste dient Verfolger- und Diagnosezwecken. Er wird ausschließlich zur Protokollierung verwendet.

  In *t_cclist* liefert CMX einen Pointer auf eine CC-Liste der Adress-Formate, für die diese Anmeldung erfolgreich war. Die CC-Liste enthält max. 16 Einträge. Die Bedeutung ist <cmx.h> zu entnehmen.

t_hostname
: Dieses Feld wird für TS-Anwendungen benötigt, die in einen Cluster-Betrieb eingebunden sind.

  Die IP-Adresse des entsprechenden IP-Interface wird als „from-address“ benutzt, wenn mittels *t_conrq()* aktiv Verbindungen aufgebaut werden.
  Bei einer Anmeldung für passiven Verbindungsaufbau werden nur solche Verbindungsanforderungen angezeigt, die über dieses IP-Interface angekommen sind.

  Dieser Parameter ist nur dann wirksam, wenn Sie im LOKALEN NAMEN der TS-Anwendung die Adress-Formate RFC1006 oder LANINET angegeben haben. Bezüglich anderer Adress-Formate als RFC1006 und LANINET gibt es keine Einschränkung für die

erste Anmeldung einer TS-Anwendung und weitere Anmeldungen für dieselbe TS-Anwendung mit identischem Hostnamen. Es dürfen sich jedoch nur solche TS-Anwendungen, die ausschließlich die Adress-Formate RFC1006 und LANINET verwenden, ein weiteres Mal mit verschiedenem Hostnamen anmelden. In diesem Fall ist der Rückgabewert T_OK und **nicht** T_NOTFIRST.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Der Prozess hat sich als Erster mit diesem Namen angemeldet.

T_NOTFIRST
: Der Aufruf war erfolgreich. Der Prozess hat sich jedoch nicht als erster Prozess für diese TS-Anwendung angemeldet.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_ENOENT
: Alle bereitgestellten Ressourcen sind belegt.

T_EFAULT
: Mindestens einer der Zeiger *name* oder *opt* (!= NULL) zeigt nicht in den Prozess-Adressraum.

T_WPARAMETER
: Der mit Hilfe von *name* übergebene LOKALE NAME oder eine der in *opt* angegebenen Optionen hat ein falsches Format oder enthält unzulässige Werte.

T_WAPPLICATION
: Der Prozess ist bereits für die TS-Anwendung angemeldet, die in *name* angegeben wurde oder der LOKALE NAME, der in *name* angegeben wurde, wird bereits von einem XTI-Prozess benutzt.

T_WAPP_LIMIT
: Der Prozess hat sich bereits für alle Anwendungen angemeldet, die ihm verfügbar sind, oder das Maximum an möglichen TS-Anwendungen ist ausgeschöpft.

T_WPROC_LIMIT
: Das Maximum an Prozessen, die CMX verwenden können, ist ausgeschöpft.

T_NOCCP
: Zum LOKALEN NAMEN, der in *name* angegeben wurde, ist (momentan) kein geeignetes CCP verfügbar.

T_WLIBVERSION
: Die Version der CMX-Bibliothek, die im Prozess eingebunden wurde, ist mit der Betriebssystemversion unverträglich.

Siehe auch

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

8.6.2 t_callback - Rückrufroutine registrieren (callback)

Mit *t_callback* erhält eine Anwendung die Möglichkeit, bei ICMX(L) eine eigene Funktion zur Ereignisbehandlung anzumelden, die dann während der Ausführung von *t_event* aufgerufen wird. Da eine ICMX(L)-Funktion (*t_event()*) diese zur Anwendung gehörende Funktion aufruft, wird diese Funktion als Rückrufroutine bezeichnet.

Die Rückrufroutine gibt der Anwendung die Möglichkeit, mit *t_event()* nicht nur auf TS-Ereignisse zu warten, sondern zeitgleich auch auf andere Ereignisse, die für die Anwendung von Belang sind (z.B. Terminal I/O). Voraussetzung dafür ist, dass sich die Ereignisquellen über Dateikennzahlen (file descriptors) beschreiben lassen, so dass die Rückrufroutine die Systemfunktionen *select()* oder *poll()* (siehe Solaris Referenzhandbuch für Programmierer) zur Ereignisprüfung verwenden kann.

Die Rückrufroutine wird im Einzelnen wie folgt verwendet. Die Funktion *t_event()* übergibt beim Aufruf an *routine* über Bitlisten Dateikennzahlen, die von ICMX(L) intern verwendet werden und für die Ereignisse ausstehen. Die Übergabe erfolgt wie beim Aufruf von *select()*. Die Funktionsroutine kann nun anwendungsspezifische Dateikennzahlen hinzufügen und mit den komplettierten Bitlisten die Funktion *select()* aufrufen. Damit ist es möglich, gleichzeitig auf TS-Ereignisse und andere anwendungsspezifische Ereignisse zu warten. Tritt ein Ereignis ein, muss *routine* überprüfen, ob es sich um ein anwendungsspezifisches Ereignis gehandelt hat oder nicht. Ist dies der Fall, beendet sich *routine* mit T_USEREVENT, ansonsten mit T_TSEVENT. Bevor sich *routine* beendet, kann die Routine ein anwendungsspezifisches Ereignis weiterverarbeiten. *routine* darf jedoch nie in Wartezustände eintreten, die nicht durch Signale unterbrechbar sind.

Das Verhalten von *t_event* richtet sich nach dem Rückkehrwert von *routine*. Ist kein TS-Ereignis eingetreten, kehrt *t_event()* mit T_NOEVENT zur Anwendung zurück. Andernfalls wird das TS-Ereignis angezeigt.

Eine Rückkehrroutine gibt der Anwendung die Möglichkeit, die Ereignisbehandlung nach eigenen Kriterien zu optimieren. Eine fehlerhafte Implementierung führt jedoch dazu, dass die Behandlung von TS-Ereignissen durch *t_event()* nicht mehr zuverlässig durchgeführt werden kann. Beachten Sie daher genau die im Folgenden angegebenen Implementierungsvorschriften.

Das Funktionsmuster von *routine* ähnelt der Funktion *select()* und sieht wie folgt aus:

```
#include <sys/select.h>
#include <cmx.h>
int   (*t_cbtype) routine (int fdsetsize,
                           fd_set *rfds,
                           fd_set *wfds,
                           fd_set *xfds,
                           struct timeval *time,
                           const void *usr);
```

-> fdsetsize
: Anzahl der Dateikennzahlen, die von ICMX(L) intern verwendet werden.

<> rfds
: Zeiger auf eine Bitliste von Dateikennzahlen, für die ICMX(L) intern Leseereignisse erwartet. Bei der Rückgabe muss die Liste mindestens die Dateikennzahlen der ICMX(L) enthalten, für die ein Leseereignis eingetreten ist. Nichtbetroffene Dateikennzahlen in der Liste werden ignoriert.

<> wfds
: Zeiger auf eine Bitliste von Dateikennzahlen, für die ICMX(L) intern die Schreiberlaubnis erwartet. Bei der Rückgabe muss die Liste mindestens die Dateikennzahlen der ICMX(L) enthalten, für die ein solches Ereignis eingetreten ist. Nichtbetroffene Dateikennzahlen in der Liste werden ignoriert.

<> xfds
: Zeiger auf eine Bitliste von Dateikennzahlen, für die ICMX(L) das Eintreten von Ausnahmebedingungen (exceptional conditions) erwartet. Bei der Rückgabe muss die Liste mindestens die Dateikennzahlen der ICMX(L) enthalten, für die ein solches Ereignis eingetreten ist. Nichtbetroffene Dateikennzahlen in der Liste werden ignoriert.

-> time
: gibt die Zeit an, die längstens auf das Eintreffen eines Ereignisses gewartet werden soll. Der Wert 0 zeigt an, dass *routine* die Existenz eines Ereignisses prüfen, aber keinen Wartezustand einnehmen darf. Der Wert -1 bedeutet, dass *routine* unbegrenzt auf das Ereignis warten soll. Der Wert von *time* folgt aus dem Wert von *t_timeout* beim Aufruf von *t_event()*.

Beachten Sie: *routine* muss den Wert 0 auf jeden Fall unterstützen, auch wenn die Anwendung *t_event()* nie im Modus T_CHECK aufruft.

-> usr

Zeiger, der von der Anwendung beim Anmelden von *routinr* an *t_callback* übergeben wurde (s. u.). Der Inhalt wird von ICMX(L) nicht überprüft. Die Anwendung kann über *usr* anwendungsspezifische Informationen an *routine* übergeben lassen.

Rückgabewert

T_NOEVENT

Es ist weder ein Benutzer- noch ein TS-Ereignis innerhalb von *time* aufgetreten. Die Funktion wurde durch ein Signal unterbrochen, oder ein interner Fehler ist aufgetreten.

t_event() beendet sich dann ebenfalls mit T_NOEVENT.

T_TSEVENT

Ein TS-Ereignis ist eingetreten. *t_event()* prüft daraufhin, welches TS-Ereignis eingetreten ist, und beendet sich mit diesem Ereignis.

Kann *t_event()* kein TS-Ereignis finden, beendet sich *t_event()* mit T_NOEVENT.

T_USEREVENT

Ein anwendungsspezifisches Ereignis ist eingetreten. *t_event()* beendet sich entweder mit T_NOEVENT, wenn *t_event()* kein TS-Ereignis bekannt ist, oder meldet das TS-Ereignis, das *t_event()* außerhalb von *routine* bekannt wurde.

Fehler

Muss *routine* aufgrund eines internen Fehlers vorzeitig abbrechen, muss sie sich mit T_NOEVENT beenden. Sie sollten den Fehlerstatus intern vermerken, damit die Anwendung nach Beendigung von *t_event()* geeignete Maßnahmen ergreifen kann.

Implementierungshinweise

– Die ICMX(L)-spezifischen Dateikennzahlen dürfen von *routine* nicht verfälscht werden. TS-Ereignisse lassen sich nur nachweisen, wenn diese Dateikennzahlen an *select()* oder *poll()* übergeben werden. Werden diese Dateikennzahlen beim Ablauf von *routine* verfälscht, arbeitet *t_event()* nicht mehr zuverlässig.

– Die Rückrufroutine darf nur Wartezustände einnehmen, die von allen Signalen unterbrochen werden können.

– Die Rückrufroutine darf nicht *t_event()* aufrufen. *t_event()* weist den rekursiven Aufruf mit T_CBRECURSIVE zurück.

– Die Rückrufroutine ist Solaris-spezifisch, da sie explizit Dateikennzahlen verwendet. Das Konzept wird daher in ICMX(L)-Implementierungen in BS2000/OSD und MS-DOS nicht angeboten.

Die Rückrufroutine wird ICMX(L) mit *t_callback* bekanntgemacht:

```
#include <cmx.h>
t_cbtype   t_callback (t_cbtype routine,
                       const void *usr,
                       const void *opt);
```

-> routine

Zeiger auf die Rückrufroutine, die von *t_event()* aufgerufen werden soll. Mit NULL wird eine bei ICMX(L) angemeldete Rückrufroutine wieder abgemeldet.

-> usr

Zeiger auf einen anwendungsspezifischen Datenbereich, der von ICMX(L) nicht überprüft wird. Der Zeiger wird von *t_event()* der Rückrufroutine beim Aufruf übergeben.

-> opt

Reserviert für spätere Erweiterungen. Der Wert muss NULL sein.

Rückgabewert

T_OK
: Der Zeiger auf die alte Rückrufroutine wird zurückgeliefert. „NULL“ als Rückgabewert bedeutet, dass keine Rückrufroutine eingehängt war.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_WPARAMETER
: *opt* ist nicht NULL.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet.

Siehe auch

t_event()

8.6.3 t_concf - Verbindung herstellen (connect confirmation)

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

t_concf() liefert:

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

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

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

– Normaldaten und (falls vereinbart) Vorrangdaten senden oder

– durch *t_event()* anzeigen, dass sie bereit ist Normaldaten bzw. (falls vereinbart) Vorrangdaten zu empfangen, die Verbindung umlenken oder abbauen.

```
#include <cmx.h>
int   t_concf (const int *tref,
               t_opt1 *opt);
```

-> tref

Zeiger zu einem Feld mit der Transportreferenz der Verbindung, die dem laufenden Prozess beim *t_event()* übergeben wurde.

<> opt

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

Damit werden die Benutzerdaten entgegengenommen, die die gerufene TS-Anwendung bei der Antwort auf den Verbindungswunsch mitgegeben hat.

Wird *opt* = NULL angegeben, so vernichtet CMX die Benutzerdaten und Optionen.

Hat die gerufene TS-Anwendung keine Benutzerdaten und Optionen angegeben, so setzt CMX die angegebenen Standardwerte.

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

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

t_optnr
: Optionsnummer. Anzugeben ist T_OPTC1.

t_udatap
: Zeiger auf einen Datenbereich, in den CMX die empfangenen Benutzerdaten der gerufenen TS-Anwendung überträgt.

  Standardwert bei Angabe von *opt* = NULL: undefiniert

t_udatal
: Vor dem Aufruf muss hier 0 oder die Länge des Datenbereiches *t_udatap* stehen. Der Bereich muss so groß sein, dass die empfangenen Daten ganz hineinpassen. Die maximal erforderliche Länge hängt vom verwendeten Transportsystem ab.

  T_MSG_SIZE ist die für alle Transportsysteme geeignete Maximalgröße. T_MSG_SIZE ist in <*cmx.h*> definiert. Nach dem Aufruf liefert CMX in diesem Feld die Anzahl der Byte zurück, die nach *t_udatap* übertragen wurden.

  Standardwert bei Angabe von *opt* = NULL: 0.

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

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

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

    Standardwert bei Angabe von *opt* = NULL: T_NO.

t_timeout
: Der Inhalt dieses Feldes ist stets T_NO.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Mindestens einer der Zeiger *opt* (!= NULL) oder *t_udatap* (!= NULL und *t_ndatal* != 0) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder auf der durch *tref* angegebenen Verbindung wurde kein T_CONCF angezeigt.

T_WPARAMETER
: Die in *opt* angegebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte oder der Puffer für die zu empfangenen Daten ist zu klein.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

t_conrq(), t_error(), t_event()

8.6.4 t_conin - Verbindungswunsch entgegennehmen (connect indication)

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

Der Aufruf liefert:

– die TRANSPORTADRESSE der rufenden TS-Anwendung,

– den LOKALEN NAMEN der lokalen TS-Anwendung,

– die Benutzerdaten, die die rufende TS-Anwendung mitgegeben hat.

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

```
#include <cmx.h>
int   t_conin (const int *tref,
               union t_address *toaddr,
               union t_address *fromaddr,
               t_opt1 *opt);
```

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

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

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

Hinweis

Erhält man bei der Kommunikation über TCP/IP mit RFC1006 eine TRANSPORTADRESSE vom Typ RFC1006, so darf diese Adresse nicht binär in einem nachfolgenden aktiven Verbindungsaufbau an *t_conrq()* übergeben

werden, da interne Adress-Bestandteile fehlen, die von *t_conrq()* ausgewertet werden. Die Anwendung muss mit *t_getname()* den GLOBALEN NAMEN erfragen und sich dann mit *t_getaddr()* die Adresse erneut beschaffen.

<> opt

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

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

Folgende Struktur ist in <cmx.h> definiert:

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

t_optnr

Optionsnummer. Anzugeben ist T_OPTC1.

t_udatap

Zeiger zu einem Datenbereich, in den CMX die empfangenen Benutzerdaten der rufenden TS-Anwendung überträgt.

Standardwert bei Angabe von *opt* = NULL: undefiniert

t_udatal

Vor dem Aufruf muss hier 0 oder die Länge des Datenbereiches *t_udatap* stehen.

Der Bereich muss dann so groß sein, dass die empfangenen Daten ganz hineinpassen. Die maximal erforderliche Länge hängt vom verwendeten Transportsystem ab. T_MSG_SIZE ist eine für alle Transportsysteme geeignete Maximalgröße. T_MSG_SIZE ist in <*cmx.h*> definiert. Nach dem Aufruf liefert CMX in diesem Feld die Anzahl der Byte zurück, die nach *t_udatap* übertragen wurden.

Standardwert bei Angabe von *opt* = NULL: 0.

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

  Dabei bedeutet:

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

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

Schlägt die rufende TS-Anwendung den Austausch von Vorrangdaten vor (T_YES), so ist die Antwort des laufenden Prozesses beim folgenden *t_conrs()* endgültig.

Wünscht die rufende TS-Anwendung keine Vorrangdaten (T_NO), können vom laufenden Prozess beim folgenden *t_conrs()* auch keine verlangt werden. Gegebenenfalls muss der laufende Prozess den Verbindungswunsch dann mit *t_disrq()* abgelehnen.

Standardwert bei Angabe von *opt* = NULL: T_NO.

t_timeout
: Der Inhalt dieses Feldes ist stets T_NO.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Mindestens einer der Zeiger *toaddr, fromaddr, opt* (!= NULL) oder *t_udatap* (!= NULL und *t_ndatal* != 0) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder auf der durch *tref* angegebenen Verbindung wurde kein T_CONIN angezeigt.

T_WPARAMETER
: Die in *opt* angegebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte oder der Puffer für die zu empfangenen Daten ist zu klein.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.5 t_conrq - Verbindung anfordern (connection request)

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

Die Verbindungsanforderung *t_conrq()* bewirkt im Einzelnen:

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

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

```
#include <cmx.h>
int   t_conrq (int *tref,
               const union t_address *toaddr,
               const union t_address *fromaddr,
               const t_opt13 *opt);
```

<- tref

Zeiger zu einem Feld, in dem CMX die verbindungsspezifische Transportreferenz zurückliefert. Sie kennzeichnet die Verbindung in den folgenden Kommunikationsphasen eindeutig. Sie muss deshalb bei allen Aufrufen, die diese Verbindung betreffen, angegeben werden.

-> toaddr

Zeiger zu einer Union *t_address* mit der TRANSPORTADRESSE der gerufenen TS-Anwendung. Die TRANSPORTADRESSE liefert der TNS als Eigenschaft zum GLOBALEN NAMEN der gerufenen TS-Anwendung. Sie kann zuvor mit Hilfe des Aufrufs *t_getaddr()* vom TNS ermittelt werden.

-> fromaddr

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

-> opt

Für *opt* ist der Wert NULL oder der Zeiger zu einer Union mit Systemoptionen anzugeben. Damit werden die Benutzerdaten und Optionen angegeben, die die gerufene TS-Anwendung mit der Verbindungsaufbauanzeige erhalten soll.

Wird *opt* = NULL angegeben, so setzt CMX die angegebenen Standardwerte.

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

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

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

t_optnr

Optionsnummer. Anzugeben ist:

T_OPTC1 bei *t_optc1*

T_OPTC3 bei *t_optc3*

t_udatap

Zeiger zu einem Bereich mit Benutzerdaten, die die gerufene TS-Anwendung mit der Verbindungsaufbauanzeige erhalten soll. Standardwert bei Angabe von *opt* = NULL: undefiniert

t_udatal

Länge der Benutzerdaten in Byte, die aus dem Bereich *t_udatap* zu übertragen sind. Wird für *t_udatal* 0 angegeben, so wird *t_udatap* nicht ausgewertet. Der Maximalwert für *t_udatal* ist abhängig vom Transportsystem (siehe Freigabemitteilung).

Standardwert bei Angabe von *opt* = NULL: 0

t_xdata

Im Parameter *t_xdata* teilt der laufende Prozess der gerufenen TS-Anwendung mit, ob er bereit ist, Vorrangdaten auszutauschen oder nicht. Zulässige Werte sind:

T_YES

Der Austausch von Vorrangdaten wird vorgeschlagen.

T_NO

Der Austausch von Vorrangdaten wird ausgeschlossen. Standardwert bei Angabe von *opt* = NULL: T_NO.

t_timeout

Für *t_timeout* ist die Inaktiv-Zeit der Verbindung anzugeben. Die Inaktiv-Zeit gibt an, wie lange die Verbindung inaktiv sein darf, bevor sie von CMX abgebaut wird.

Sie beginnt erst ab dem Zeitpunkt, ab dem alle Daten abgeholt wurden.

Mögliche Angaben:

T_NO

Die Inaktiv-Zeit der Verbindung wird nicht überwacht.

n > 0

Die Verbindung darf n Sekunden inaktiv sein. Danach baut CMX sie ab.

Standardwert bei Angabe von *opt* = NULL: T_NO.

t_ucepid

In diesem Feld kann eine beliebige Benutzerreferenz dieser Verbindung an CMX übergeben werden. Diese Benutzerreferenz kann dem laufenden Prozess als Option beim *t_event()* von CMX zurückgeliefert werden. Damit kann der laufende Prozess, falls er mehrere Verbindungen hält, Ereignisse der entsprechenden Ver-

bindung über ein selbst gewähltes Merkmal zuordnen. Die Benutzerreferenz ist eine Alternative zu der von CMX gewählten Transportreferenz *tref*.

Standardwert bei Angabe von *opt* = NULL: 0

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Mindestens einer der Zeiger *toaddr, fromaddr, opt* (!= NULL) oder *t_udatap* (!= NULL und *t_ndatal* != 0) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder der Prozess hat in der lokalen TS-Anwendung, die in *fromaddr* angegeben ist, in *t_apmode* nicht T_ACTIVE gesetzt.

T_WPARAMETER
: Die durch *toaddr* übergebene TRANSPORTADRESSE oder der durch *fromaddr* übergebene LOKALE NAME oder eine der in *opt* übergebenen Optionen hat ein falsches Format oder enthält unzulässige Werte.

T_WAPPLICATION
: Der Prozess ist nicht in der TS-Anwendung angemeldet, die den durch *fromaddr* übergebenen LOKALEN NAMEN hat.

T_WCONN_LIMIT
: Der Prozess hat die Anzahl an Verbindungen bereits ausgeschöpft, die für diese TS-Anwendung beim *t_attach()* (Parameter *t_conlim*) vereinbart wurden oder der Systemgrenzwert für Verbindungen ist überschritten.

T_NOCCP
: Die in *toaddr* angegebene TRANSPORTADRESSSE wird von keinem (derzeit) betriebsbereiten CCP unterstützt oder der in *fromaddr* angegebene LOKALE NAME enthält für dieses CCP keine Informationen.

T_ETIMEOUT
: Das CCP antwortet nicht im Zeitlimit.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.6 t_conrs - Verbindungswunsch beantworten (connection response)

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

Mit der Antwort *t_conrs()*

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

– ist die Verbindung für den laufenden Prozess fertig aufgebaut.

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

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

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

– die Verbindung abbauen oder umlenken.

```
#include <cmx.h>
int   t_conrs (const int *tref,
               const t_opt13 *opt);
```

-> tref

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

-> opt

Für *opt* ist der Wert NULL oder der Zeiger zu einer Union mit Systemoptionen anzugeben.

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

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

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

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

t_optnr

Optionsnummer. Anzugeben ist:

T_OPTC1 bei *t_optc1*,

T_OPTC3 bei *t_optc3*.

t_udatap

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

Standardwert bei Angabe von *opt* = NULL: undefiniert

t_udatal

Länge der aus dem Bereich *t_udatap* zu übertragenden Benutzerdaten in Byte. Wird für *t_udatal* 0 angegeben, so wird *t_udatap* nicht ausgewertet. Der Maximalwert für *t_udatal* ist abhängig vom Transportsystem (siehe Freigabemitteilung).

Standardwert bei Angabe von *opt* = NULL: 0

t_xdata

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

Zulässige Werte sind:

T_YES

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

T_NO

Vorrangdaten werden abgelehnt.

Die Antwort ist verbindlich.

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

Standardwert bei Angabe von *opt* = NULL: T_NO.

t_timeout

Für *t_timeout* ist die Inaktiv-Zeit der Verbindung anzugeben. Die Inaktiv-Zeit gibt an, wie lange die Verbindung inaktiv sein darf, bevor sie von CMX abgebaut wird.

T_NO

Inaktivzeit wird nicht überwacht.

n > 0

Die Verbindung darf n Sekunden inaktiv sein. Danach baut CMX sie ab.

Standardwert bei Angabe von *opt* = NULL: T_NO.

t_ucepid

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

Standardwert bei Angabe von *opt* = NULL: 0

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Mindestens einer der Zeiger *opt* (!= NULL) oder

  t_udatap (!= NULL und *t_ndatal* != 0) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder es ging kein erfolgreicher Aufruf *t_conin()* voraus.

T_WPARAMETER
: Die durch *opt* übergebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte.

T_COLLISION
: Für die Verbindung liegt das Ereignis T_DISIN (Verbindungsabbauanzeige) vor, wurde aber noch nicht mit *t_event()* abgefragt.

  Reaktion: *t_event()* aufrufen.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

t_conin(), t_error(), t_event()

8.6.7 t_datago - Datenfluss freigeben (data go)

t_datago() gibt die gesperrte Datenanzeige auf der angegebenen Verbindung frei. Der laufende Prozess teilt CMX dadurch mit, dass er wieder bereit ist, Daten entgegenzunehmen. Dieser Aufruf gibt auch die Vorrangdatenanzeige (sofern vereinbart) wieder frei, falls sie (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 (im Verlauf) das Ereignis T_DATAGO zugestellt, sie darf wieder Daten senden.

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

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

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

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS kann auftreten:

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder der Prozess ist für die in *tref* angegebene Verbindung nicht in der Datenphase oder die Anzeige von Daten ist nicht gesperrt.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.8 t_datain - Daten empfangen (data indication)

t_datain() übernimmt ein zuvor mit *t_event()* angezeigtes Ereignis T_DATAIN. *t_datain()* muss dabei vor dem nächsten *t_event()* aufgerufen werden.

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

Die maximale Länge der TIDU ist abhängig vom verwendeten Transportsystem. Sie kann für eine bereits aufgebaute Verbindung mit *t_info()* abgefragt werden.

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

Zwischen zwei TIDUs derselben TSDU können beliebige andere CMX-Ereignisse für dieselbe oder eine andere Verbindung auftreten.

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

t_datain() zeigt an:

– (im Parameter *chain*)

  ob noch eine weitere TIDU zur laufenden TSDU gehört (*chain* = T_MORE) oder nicht (*chain* = T_END).

  Die einzelnen TIDUs einer TSDU werden jeweils bei *t_event()* mit dem Ereignis T_DATAIN angezeigt.

– (mit dem Ergebniswert)

  ob die aktuelle TIDU vollständig gelesen wurde oder nicht.

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

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

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

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

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

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

  Mögliche Werte:

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

  T_END
  : Die vorliegende TIDU ist die letzte der TSDU.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Die TIDU wurde vollständig gelesen.

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

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Der Zeiger *datap* zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder für die in *tref* angezeigte Verbindung wurde kein T_DATAIN angezeigt.

T_WPARAMETER
: Die in *datal* angegebenen Länge ist unzulässig.

T_COLLISION
: Für die Verbindung liegt das Ereignis T_DISIN (Verbindungsabbauanzeige) vor, wurde aber noch nicht mit *t_event()* abgefragt.

  Reaktion: *t_event()* aufrufen.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.9 t_datarq - Daten senden (data request)

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

Die TIDU, die von *t_datarq()* übertragen werden soll, muss vom laufenden Prozess in einem zusammenhängenden Datenbereich bereitgestellt werden.

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

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

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

Der Fluss der TIDUs kann gesperrt werden:

– von der empfangenden TS-Anwendung. Sie kann den Fluss der TIDUs durch Aufruf von *t_datastop()* oder *t_xdatstop()* sperren.

– von CMX, wenn der lokale Zwischenspeicher voll ist.

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

Ein erfolgreicher Abschluss von *t_datarq()* (T_OK) bedeutet nicht, dass die empfangende TS-Anwendung die Daten bereits entgegengenommen hat.

Ein erfolgloser Abschluss von *t_datarq()* (T_ERROR) bedeutet stets, dass lokal ein Fehler erkannt wurde.

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

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

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

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

-> chain
: Zeiger zu einem Indikator, durch den der Prozess anzeigt, ob noch eine weitere TIDU zur TSDU gehören.

  Mögliche Werte:

  T_MORE
  : Es folgt noch eine weitere TIDU, die zur TSDU gehört.

  T_END
  : Die vorliegende TIDU ist die letzte der TSDU.

Rückgabewert

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

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

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Der Zeiger *datap* zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder der Prozess ist für die in *tref* angegebene Verbindung nicht in der Datenphase oder der Datenfluss ist gesperrt.

T_WPARAMETER
: Die in *datal* angegebene Länge oder der in *chain* angegebene Wert ist unzulässig.

T_COLLISION
: Für die Verbindung liegt das Ereignis T_DISIN (Verbindungsabbauanzeige) vor, wurde aber noch nicht mit *t_event()* abgefragt.

  Reaktion: *t_event()* aufrufen.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.10 t_datastop - Datenanzeige sperren (data stop)

t_datastop() sperrt die Datenanzeige auf der angegebenen Verbindung.

t_datastop() bewirkt im Einzelnen:

– Der laufende Prozess teilt CMX so mit, dass er bis auf weiteres nicht bereit ist, für diese Verbindung Daten zu empfangen. Ein bereits angezeigtes Ereignis T_DATAIN muss aber erst beantwortet werden.

– Der laufende Prozess bekommt das Ereignis T_DATAIN für die angegebene Verbindung nicht mehr zugestellt. Er kann aber, während die Datenanzeige gesperrt ist, andere CMX-Funktionen aufrufen, z. B. eine weitere Verbindung aufbauen, abbauen oder umlenken.

– Die sendende TS-Anwendung erhält (im Verlauf bei *t_datarq()* das Ergebnis T_DATASTOP. Sie darf keine Daten mehr senden (siehe auch Abschnitt „Transportsystem-spezifische Besonderheiten“ auf Seite 104).

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

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

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

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder der Prozess ist für die in *tref* angegebene Verbindung nicht in der Datenphase oder eine TIDU oder eine ETSDU ist noch nicht komplett gelesen.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.11 t_detach - Abmelden Prozess aus TS-Anwendung (detach process)

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

Meldet sich der letzte Prozess einer TS-Anwendung ab, so wird die TS-Anwendung aufgelöst. Verbindungswünsche für diese TS-Anwendung werden dann nicht mehr angenommen.

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

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

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Der Zeiger *name* zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet.

T_WPARAMETER
: Der durch *name* übergebene LOKALE NAME hat ein falsches Format oder enthält unzulässige Werte.

T_WAPPLICATION
: Der Prozess ist nicht in der TS-Anwendung angemeldet, die den durch *name* übergebenen LOKALEN NAMEN hat.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

t_attach(), t_error()

8.6.12 t_disin - Verbindungsabbau entgegennehmen (disconnection indication)

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

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

t_disin() liefert ferner:

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

– den Grund für den Abbau der Transportverbindung. Die Klartextdarstellung des Codes erhält man mit Hilfe von *t_preason()* bzw. *t_strreason()*.

```
#include <cmx.h>
int   t_disin (const int *tref,
               int *reason,
               t_opt2 *opt);
```

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

<- reason
: Zeiger zu einem Feld, in das CMX den Grund des Verbindungsabbaus einträgt.

  Mögliche Werte:

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

  sonst:
  : Die Verbindung wurde durch CMX oder das Transportsystem abgebaut.

  Die möglichen Werte für diesen Parameter und deren Bedeutung finden Sie im Anhang dieses Handbuchs. Der von CMX gelieferte Code für den Verbindungsabbau kann mit Hilfe des Kommandos *cmxdec* (siehe CMX „Betrieb und Administration“ [1]) decodiert werden.

<> opt
: Für *opt* ist der Wert NULL oder der Zeiger auf eine Union anzugeben, die eine Struktur mit Systemoptionen enthält.

Damit können die Benutzerdaten abgefragt werden, die die ferne TS-Anwendung beim Verbindungsabbau angegeben hat.

Wird *opt* = NULL angegeben, so vernichtet CMX die Benutzerdaten und Optionen.

Hat die ferne TS-Anwendung keine Benutzerdaten und Optionen angegeben, so liefert CMX die angegebenen Standardwerte.

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

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

t_optnr
: Optionsnummer. Anzugeben ist T_OPTC2.

t_udatap
: Zeiger zu einem Datenbereich, in den CMX die empfangenen Benutzerdaten der fernen TS-Anwendung überträgt.

  Standardwert bei Angabe von *opt* = NULL: undefiniert

t_udatal
: Vor dem Aufruf muss hier 0 oder die Länge des Datenbereiches *t_udatap* stehen.

  Der Bereich muss dann so groß sein, dass die empfangenen Daten ganz hineinpassen. Die maximal zulässige Länge der Benutzerdaten hängt vom verwendeten Transportsystem ab. T_MSG_SIZE ist eine für alle Transportsysteme geeignete Maximalgröße. Nach dem Aufruf liefert CMX in diesem Feld die Anzahl der nach *t_udatap* übertragenen Byte zurück.

  Standardwert bei Angabe von *opt* = NULL: 0.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Der Zeiger *opt* (!= NULL) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder für die in *tref* angegebene Verbindung wurde kein T_DISIN angezeigt.

T_WPARAMETER
: Die in *opt* angegebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte oder der Puffer für die zu empfangenen Daten ist zu klein.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.13 t_disrq - Verbindung abbauen (disconnection request)

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

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

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

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

```
#include <cmx.h>
int   t_disrq (const int *tref,
               const t_opt2 *opt);
```

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

-> opt
: Für *opt* ist der Wert NULL oder der Zeiger auf eine Union anzugeben, die eine Struktur mit Systemoptionen enthält. Damit werden die Benutzerdaten angegeben, die die ferne TS-Anwendung mit der Anzeige des Verbindungsabbaus erhalten soll.

  Wird *opt* = NULL angegeben, so nimmt CMX die angegebenen Standardwerte.

  Folgende Struktur ist in <cmx.h> definiert:

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

  t_optnr
  : Optionsnummer. Anzugeben ist T_OPTC2.

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

Standardwert bei Angabe von *opt* = NULL: undefiniert

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

  Wird *t_udatal* = 0 angegeben, so wird *t_udatap* nicht ausgewertet. Der Maximalwert für *t_udatal* ist abhängig vom Transportsystem (siehe Freigabemitteilung).

  Standardwert bei Angabe von *opt* = NULL: 0

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Mindestens einer der Zeiger *opt* (!= NULL) oder *t_udatap* (!= NULL und *t_ndatal* != 0) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder die in *tref* angegebene Verbindung ist weder aufgebaut noch im Aufbau, noch wird sie umgelenkt.

T_WPARAMETER
: Die durch *opt* übergebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.14 t_error - Fehlerdiagnose (error)

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

Die möglichen Fehlermeldungen zu den Aufrufen an der Programmschnittstelle ICMX(L) entstehen entweder in den Funktionen der CMX-Bibliothek im Benutzerprozess oder im Betriebssystemkern. Sie müssen danach unterschieden werden, ob sie in CMX selbst erzeugt werden oder aus Betriebssystemaufrufen in CMX resultieren. Falls Ersteres zutrifft, liefert *t_error()* einen CMX-internen Fehlercode, falls Letzteres zutrifft, den Inhalt der externen Variablen *errno*.

In beiden Fällen kann der Fehlercode mit Hilfe der Aufrufe *t_strerror()* bzw. *t_perror()* in Klartext übersetzt werden. *t_strerror()* liefert einen Zeiger auf einen statischen Bereich, der die Klartextdarstellung der Fehlermeldung enthält.

t_perror() schreibt den Klartext der Fehlermeldung nach *stderr*. Der Fehlercode kann auch mit Hilfe des Kommandos *cmxdec* (siehe CMX „Betrieb und Administration" [1]) decodiert werden. Der Aufbau der Fehlermeldungen von CMX ist im Abschnitt „Fehlerbehandlung“ auf Seite 40 beschrieben.

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

Rückgabewert

Rückgabewert von *t_error()* ist der hexadezimale Code des von CMX erzeugten Fehlerwerts. Die Fehlerwerte sind in *<cmx.h>* definiert. Eine Liste aller möglichen Fehlerwerte mit Fehlertyp *T_CMXTYPE (0)* und Fehlerklasse *T_CMXCLASS (0)*, d. h. alle möglichen Rückgabewerte von *t_error()* finden Sie im Anhang.

Bei der Beschreibung der einzelnen Funktionsaufrufe an ICMX(L) sind unter der Überschrift 'Fehler' jeweils die Fehlerwerte aufgelistet, die *t_error()* im Fall eines Abbruchs der jeweiligen Funktion zurückliefert.

Dateien

<cmx.h> – globale CMX-Definitionsdatei
<tnsx.h> – TNS-Definitionsdatei
<errno.h> – Meldungen zu Systemaufrufen

Siehe auch

t_perror(), t_strerror()

8.6.15 t_event - Ereignis abwarten oder abfragen (event)

t_event() stellt fest, ob ein CMX-Ereignis für den laufenden Prozess eingetroffen ist.

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

– **synchron** darauf warten, dass ein CMX-Ereignis für den laufenden Prozess eintrifft. Der Prozess wird während dieser Wartezeit suspendiert. Der Wartezustand kann durch Signale unterbrochen werden. In den Optionen *opt* kann man eine Zeitschranke für das synchrone Warten angeben. Ist innerhalb dieser Wartezeit kein Ereignis eingetroffen, wird der Wartezustand abgebrochen.

– **asynchron** prüfen, ob ein CMX-Ereignis für den laufenden Prozess vorliegt. Die Funktion kehrt immer sofort zum laufenden Prozess zurück.

Neben dem entsprechenden Ereignis liefert *t_event()*:

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

– ereignisspezifische Zusatzinformationen, falls diese in den Optionen *opt* vereinbart wurden.

t_event() erlaubt CMX außerdem, weitere Daten für eine Verbindung anzuzeigen, wenn die Datenanzeige nicht explizit für diese Verbindung durch *t_datastop()* bzw. *t_xdatstop()* gesperrt wurde.

Ist für einen Prozess ein Ereignis T_DATAIN oder T_XDATIN angezeigt, so darf z. B. die betroffene Verbindung nicht umgelenkt werden (siehe Abschnitt „Zustände von TS-Anwendungen und Zustandsübergänge“ auf Seite 97). Insbesondere darf *t_event()* erst wieder aufgerufen werden, wenn der laufende Prozess die damit angezeigten Daten mit *t_datain()*, *t_vdatain()* bzw. *t_xdatin()* entgegengenommen hat.

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

Ausnahmen

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

– Das Ereignis T_DISIN (Verbindungsabbau-Anzeige) kann die Ereignisse T_DATAIN und T_XDATIN für die betroffene Verbindung überholen und damit zerstören.

  Die Daten, die T_DATAIN bzw. T_XDATIN anzeigen sollte, gehen verloren.

```
#include <cmx.h>
int   t_event (int *tref,
               int cmode,
               t_opte *opt);
```

<- tref

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

-> cmode

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

Mögliche Werte:

T_WAIT (synchrone Verarbeitung)

Der laufende Prozess wird suspendiert, bis ein TS-Ereignis eintritt, die vereinbarte Wartezeit abläuft (Parameter *t_timeout* in *opt*) oder ein Signal (*z. B. alarm(CES)*) auftritt. In den letzten beiden Fällen wird dann das Ereignis T_NOEVENT geliefert.

T_CHECK (asynchrone Verarbeitung)

Der laufende Prozess prüft, ob ein TS-Ereignis vorliegt.

Liegt ein TS-Ereignis für den laufenden Prozess vor, so wird ihm dieses Ereignis geliefert.

Liegt kein Ereignis vor, so wird dem Prozess das Ereignis T_NOEVENT geliefert.

-> opt

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

CMX setzt bei Angabe NULL die angegebenen Standardwerte.

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

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

    struct  t_opte2 {
->     int  t_optnr;      /* Options-Nr. */
<-     int  t_attid;      /* CMX-Referenz der Anmeldung */
<-     int  t_uattid;     /* Benutzerreferenz der
                             Anmeldung */
<-     int  t_ucepid;     /* Benutzerreferenz der
                             Verbindung */
->     int  t_timeout;    /* Zeitschranke für T_WAIT */
<-     int  t_evdat;      /* ereignisspezifische
                             Informationen */
<-     int  t_evinf[10];  /* BS2000 Ereignis
                             Informationsworte */
    };
```

t_optnr

Optionsnummer. Anzugeben ist:

T_OPTE1 in *t_opte1*
T_OPTE2 in *t_opte2*

t_attid

In *t_attid* liefert *t_event()* die CMX-interne Referenz der betroffenen Anmeldung.

Die CMX-Referenz wurde als Option von CMX auch beim *t_attach()* geliefert. Sie dient nur Verfolger- und Diagnosezwecken und wird ausschließlich zur Protokollierung verwendet.

t_uattid

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

t_ucepid

In *t_ucepid* liefert *t_event()* die Benutzerreferenz der betroffenen Verbindung bei den TS-Ereignissen T_CONCF, T_DATAIN, T_XDATIN, T_DATAGO, T_XDATGO und T_DISIN.

Die Benutzerreferenz wurde bei *t_conrq()*, *t_conrs()* oder *t_redin()* an CMX übergeben. Ein Prozess, der mehrere Verbindungen hält, kann damit die TS-Ereignisse der entsprechenden Verbindung zuordnen. Dieses vom Benutzer gewählte Merkmal ist eine Alternative zur Transportreferenz *tref*, die von CMX festgelegt wird.

t_timeout

Bei *cmode* = T_WAIT:

Für *t_timeout* kann eine Wartezeit angegeben werden, während der *t_event()* synchron auf ein Ereignis warten soll.

Bei *cmode* = T_CHECK: Eine Angabe für *t_timeout* wird ignoriert.

Mögliche Angaben für *t_timeout*:

T_NOLIMIT

Es wird keine Wartezeit vorgegeben. Der Prozess wartet (unbegrenzt) so lange, bis ein Ereignis eintritt oder *t_event()* durch ein Signal abgebrochen wird.

T_NO

Der Prozess wartet nicht. Er wird sofort mit dem vorhandenen TS-Ereignis oder mit T_NOEVENT fortgesetzt (entspricht *cmode* = T_CHECK).

n > 0

Der Prozess wartet n Sekunden auf das Eintreffen eines TS-Ereignisses. Tritt in diesem Zeitraum kein TS-Ereignis für den wartenden Prozess ein, so wird er mit dem Ereignis T_NOEVENT fortgesetzt. Ein Abbruch durch Signale ist möglich.

Standardwert bei Angabe von *opt* = NULL: T_NOLIMIT

t_evdat
: Hier liefert CMX ereignisspezifische Zusatzinformationen zurück.

  Mögliche Information:

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

t_evinf[10]
: Dieses Feld wird von BS2000 Anwendungen benutzt und wird von CMX in Solaris nicht unterstützt.

Rückgabewert

T_CONIN
: Das Ereignis zeigt an, dass eine rufende TS-Anwendung eine Verbindung zum laufenden Prozess aufbauen will. Diese Verbindungsaufbauanzeige muss zunächst mit *t_conin()* abgeholt werden und dann mit *t_conrs()* bestätigt oder mit *t_disrq()* abgelehnt werden.

T_CONCF
: Dieses Ereignis zeigt an, dass die gerufene TS-Anwendung, einen Verbindungsaufbauwunsch des laufenden Prozesses positiv beantwortet hat. Diese Verbindungsaufbaubestätigung muss mit *t_concf()* abgeholt werden.

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

T_DATAGO
: Die lokale TS-Anwendung kann auf der in *tref* angegebenen Verbindung weiter Daten senden.

  Mögliche Reaktion: *t_datarq()* oder *t_vdatarq()*.

  Das Ereignis T_DATAGO erlaubt es der lokalen TS-Anwendung auch, auf dieser Verbindung wieder Vorrangdaten zu senden, sofern beim Verbindungsaufbau das Senden und Empfangen von Vorrangdaten vereinbart wurde.

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

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

T_NOEVENT
: Das Ereignis bedeutet:

  bei cmode = T_CHECK

  Es liegt kein Ereignis vor.

  bei cmode = T_WAIT

  Wartezustand des Prozesses abgebrochen. Der Abbruch erfolgte durch ein Signal oder wegen Ablauf der vereinbarten Wartezeit. Ein TS-Ereignis trat nicht auf.

  Der Inhalt von *tref* ist nicht definiert.

T_REDIN
: Dieses Ereignis zeigt an, dass ein anderer Prozess derselben TS-Anwendung eine Verbindung auf den laufenden Prozess umgelenkt hat. Die Verbindungsumlenkung muss mit *t_redin()* abgeholt werden.

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

  Dieses Ereignis wird nur angezeigt:

  – wenn beim Verbindungsaufbau der Austausch von Vorrangdaten vereinbart wurde.

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

T_XDATGO
: CMX zeigt mit diesem Ereignis an, dass der Prozess auf der in *tref* angegebenen Verbindung weiter Vorrangdaten senden darf.

  Mögliche Reaktion: *t_xdatrq()*.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Der Zeiger *opt* (!= NULL) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet, oder eine TIDU oder ETSDU wurde noch nicht komplett gelesen.

T_WPARAMETER
: Der in *cmode* angegebene Wert ist unzulässig, oder die in *opt* angegebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte.

T_CBRECURSIVE
: Rekursiver Aufruf von *t_event* in einer callback-Routine ist nicht erlaubt.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.16 t_getaddr - TRANSPORTADRESSE für den GLOBALEN NAMEN abfragen

t_getaddr() liefert entweder die TRANSPORTADRESSE eines Objekts *globname* aus dem TS-Directory 1 (*globname* !=NULL) oder ein Template (*globname* = NULL) zurück, dessen Adress-Format(e) davon abhängen, welcher Wert via *opt* übertragen wird. Für den Parameter globname ist der GLOBALE NAME der TS-Anwendung anzugeben. *t_getaddr()* liefert als Ergebnis einen Zeiger auf einen statischen Bereich mit der TRANSPORTADRESSE dieser TS-Anwendung zurück.

```
const struct t_partaddr *t_getaddr (const char *glob,
                                    const t_optg *opt);
```

globname
: ist der GLOBALE NAME der TS-Anwendung.

t_getaddr()
: liefert einen Zeiger auf einen statischen Bereich mit der TRANSPORTADRESSE dieser TS-Anwendung zurück.

„->globname !=NULL“
: bezeichnet den GLOBALEN NAMEN des Objekts im TS-Directory und wird als mit NIL endender String in der Form NP5.NP4.NP3.NP2.NP1 erwartet. Die NPi repräsentieren die Namensteile des GLOBALEN NAMENS in aufsteigender hierarchischer Ordnung von links nach rechts. Hat ein NPi keinen Wert, so muss der Separator '.' angegeben werden, wenn dieser Namensteil von mindestens einem weiteren Namensteil gefolgt wird, der in der Hierarchie höher ist (mehrere '.' am Ende können weggelassen werden). Mindestens einer der Namensteile NPi muss einen Wert haben. '.' muss im NPi als '\.' angegeben werden.

„->globname == NULL“
: bewirkt, dass *t_getaddr* einen Zeiger auf ein Template zurückgibt. Seine Struktur ist abhängig von dem Wert, der via Parameter *opt* übermittelt wird.

„->opt, wenn globname !=NULL“
: Zeiger auf eine Union mit Systemoptionen oder NULL. Bei *globname* != NULL ignoriert CMX in UNIX diesen Parameter.

  Die folgende Struktur ist in <cmx.h> nur aus Gründen der Kompatibilität zum CMX in BS2000/OSD definiert:

```
    struct t_optg1 {
->      int  t_optnr;         /*Option no. */
->      int  t_evref;         /* system event reference
                                 point */
->      char *t_buf [200];    /* workarea */
    }
```

t_optnr
: Optionsnummer. Geben Sie an: T_OPTG1 in *t_optg1*

t_evref
: Dieses Feld wird von BS2000/OSD benutzt und wird von CMX in UNIX nicht unterstützt.

t_buf [200]
: Dieses Feld wird von BS2000/OSD benutzt und wird von CMX in UNIX nicht unterstützt.

„->opt, wenn globname == NULL“

Zeiger auf eine Union, die folgende, in <cmx.h> definierte Struktur enthält:

```
    struct t_optg7 {
->      int t_optnr;       /*Option no. = T_OPTG7        */
->      int t_addrtype;    /* selected address format    */
    }
```

In diesem Fall liefert *t_getaddr* einen Zeiger auf eine Adress-Schablone zurück. Die Struktur der Adress-Schablone ist abhängig von dem Wert von *t_addrtype* in *t_optg7*:

Adress-Format CX_RFC1006, selektiert via addrtype = T_INETA

```
pa_header | rest ...........
02000027    00040010 001d300d 03490000 00070001 f0f0f0f0 fe810200 668008
            |   |    |                          |             |port|
      cx_type   ka_type/ka_size                 IPv4-Adr.    nr.|
            61 61616161 616161
            | - T-Selektor - |
```

Adress-Format CX_LANINET, selektiert via addrtype = T_INETA | T_LANINET

```
pa_header | rest ...........
0200001f    01000010 0016300d 03490000 00070001 f0f0f0f0 fe800431 313131
            |   |    |                          |             | T-Sel.
      cx_type   ka_type/ka_size                 IPv4-Adresse
```

Adress-Format CX_RFC1006, selektiert via addrtype = T_INETA6

```
pa_header | rest ...........
02000033    00040010 00293019 03490000 00130001 fe800000 00000000
            |   |    |                          |
      cx_type   ka_type/ka_size                 IPv6-Adresse
028017ff    fe287b08 fe810200 66800861 61616161 616161
                          |        |
                    Portnummer     T-Selektor
```

Adress-Format CX_LANINET, selektiert via addrtype = T_INETA6 / T_LANINET

```
pa_header | rest ...........
0200002b    01000010 00213019 03490000 00130001 fe800000 00000000
            |   |    |                          |
      cx_type   ka_type/ka_size                 IPv6-Adresse
028017ff    fe287b08 fe800431 313131
                          |
                         T-Selektor
```

Adress-Format CX_OSITYPE, selektiert via addrtype = T_OSI

```
pa_header | rest ...........
02000020    20000010 0016400a 49006cf0 f0f0f0f0 f0008008 61616161 61616161
            |   |    |              |-MAC-Adresse|       | - T-Selektor -|
      cx_type   ka_type/ka_size

            /* AU90 + AU91 (Routing/CC-Info) Adress-Format */
```

Adress-Format CX_WANNEA, selektiert via addrtype = T_NEA

```
pa_header | rest ...........
02000018    00010010 000e4102 ffaa8008 61616161 61616161
            |   |    |        | Reg-/  |- T-Selektor - |
      cx_type  ka_type/ka_size  Proz-Nr

        /* AU90 + AU91 (Routing/CC-Info) z.Zt. nicht administrierbar */
```

Adress-Format CX_WANSBKA mit ISDN-Nr., selektiert via addrtype = T_E164

```
pa_header | rest ............
0200001e    00100010 00147008 03063131 31313131 80086161 61616161 6161
            |   |    |            |ISDN-Nummer|     | - T-Selektor - |
      cx_type   ka_type/ka_size

   /* AU90 + AU91 (Routing/CC-Info) + AUD0 + AUD1 (Transportprotokoll-
   Identifier/Protokollklassen-Defaults) z.Zt. nicht administrierbar */
```

Adress-Format CX_WANSBKA mit Telefonnr., selektiert via addrtype = T_E163

```
pa_header | rest ...........
0200001e    00100010 00147008 08063131 31313131 80086161 61616161 6161
            |   |    |            |Tel.-Nummer|     | - T-Selektor - |
      cx_type   ka_type/ka_size

    /* AU90 + AU91 (Routing/CC-Info) + AUD0 + AUD1
(Transportprotokoll-    Identifier/Protokollklassen-Defaults) z.Zt. nicht
administrierbar */
```

Adress-Format CX_WANSBKA mit DTE-Adr., selektiert via addrtype = T_X121

```
pa_header | rest............
0200001c    00100010 00107008 01041111 111f8008 61616161 61616161
            |   |    |            |DTE-Adr|     | - T-Selektor - |
      cx_type   ka_type/ka_size

    /* AU90 + AU91 (Routing/CC-Info) + AUD0 + AUD1
(Transportprotokoll-   Identifier/Protokollklassen-Defaults) z. Zt. nicht
administrierbar */
```

Adress-Format CX_WAN3SBKA mit DTE-Adr., selektiert via addrtype = T_X121|T_NULLTP

```
pa_header | rest.............
0200001c    00080010 00123007 06370611 11118008 61616161 61616161
            |   |    |            |DTE-Adr|     | - T-Selektor - |
      cx_type   ka_type/ka_size

        /* AU90 + AU91 (Routing/CC-Info) z. Zt. nicht administrierbar */
```

Adress-Format CX_WANSBKA mit X.21-Rufnr., selektiert via addrtype = T_X21

```
pa_header | rest.............
0200001e    00100010 00147008 08063131 31313131 80086161 61616161 6161
            |   |    |            |Tel. Nummer|       | - T-Selektor - |
      cx_type   ka_type/ka_size

   /* AU90 + AU91 (Routing/CC-Info) + AUD0 + AUD1
(Transportprotokoll-   Identifier/Protokollklassen-Defaults) z. Zt. nicht
administrierbar */
```

Adress-Format CX_WANSBKA mit PVC-Nr., selektiert via addrtype = T_PVC

```
pa_header | rest.............
0200001a    00100010 00107004 050200ff 80086161 61616161 6161
            |   |    |            |PVC     | - T-Selektor - |
      cx_type   ka_type/ka_size    nr.

    /* AU90 + AU91 (Routing/CC-Info) + AUD0 + AUD1
(Transportprotokoll-    Identifier/Protokollklassen-Defaults) z. Zt. nicht
administrierbar */
```

Um die Werte des Templates zu verändern, verwenden Sie *t_setaddrpart*.

Rückgabewert

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

Fehler

Beim Auftreten eines Fehlers kann der Fehlercode mit *t_error()* abgefragt werden.

Es können Fehlerwerte der Fehlertypen T_DSTEMP_ERR, T_DSCALL_ERR, T_DSPERM_ERR, T_DSWARNING auftreten.

Mit Fehlerklasse T_DSPAR_ERR können folgende Fehlerwerte auftreten:

T_DIRERR
: Das TS_Directory DIR1 ist nicht vorhanden.

T_NAMERR
: Der in *globname* angegebene GLOBALE NAME existiert im TS-Directory DIR1 nicht.

T_ILLNAM
: Der in *globname* angegebene GLOBALE NAME ist syntaktisch falsch (zu viele Namensteile, ungültige Längen der Namensteile, ungültige Zeichen innerhalb des Namens.)

T_PROPER
: Dem in *globname* angegebene GLOBALE NAME ist kein LOKALER NAME zugeordnet.

Mögliche Fehlerwerte mit Fehlerklasse T_DSSYS_ERR sind die in <errno.h> definierten Systemfehlermeldungen.

Mit Fehlerklasse T_DSILL_VERS kann folgender Fehlerwert auftreten:

T_NOTSPEC
: Die im Prozess eingebundene Version der CMX-Bibliothek und die CMX-Ablaufumgebung sind unverträglich.

Mit Fehlerklasse T_DSINT_ERR sind folgende Fehlerwerte möglich:

T_TIMOUT
: Der TNS-Dämon tnsxd (CMX_1) antwortet nicht innerhalb eines Zeitraums von 20 Sekunden.

T_PROT
: Im tnsxd (CMX_1)-Protokoll treten Fehler auf.

T_LFILE
: Das TS_Directory DIR1 hat kein korrektes Format.

Verwendung

Der oben erwähnte statische Bereich wird bei jedem Aufruf überschrieben. Der Aufrufende muss den Bereich kopieren, wenn er gesichert werden soll. Die Menge der zu kopierenden Daten kann aufgrund der Größe des Feldes *t_palng* in der Struktur *t_partaddr* bestimmt werden.

8.6.17 t_getaddrpart, t_setaddrpart - Lesen oder Ändern der Adress-Information in TRANSPORTADRESSE

t_getaddrpart - Adress-Information aus TRANSPORTADRESSE lesen
t_setaddrpart - Adress-Information in TRANSPORTADRESSE ändern

t_getaddrpart() liefert die einzelnen service-spezifischen Adress-Informationen zur TRANSPORTADRESSE *addr*.

t_setaddrpart() liefert eine geänderteTRANSPORTADRESSE *newaddr*, wobei das aufrufende Programm in *addr* die ursprüngliche TRANSPORTADRESSE und in *opt* die Modifikation der einzelnen service-spezifischen Adress-Informationen angibt.

```
#include <cmx.h>
int   t_getaddrpart (const union t_address *addr,
                     t_optg *opt);
int   t_setaddrpart (const union t_address *addr,
                     union t_address *newaddr,
                     const t_optg *opt);
```

-> addr

Zeiger auf eine Union *t_address*, die eine TRANSPORTADRESSE enthält. Das Anwendungsprogramm kann diese TRANSPORTADRESSE als Rückgabewert von *t_getaddr()* oder als Parameter *fromaddr* von *t_conin()* erhalten haben.

<- newaddr

Zeiger auf eine Union *t_address*. In dieser Union liefert *t_setaddrpart()* die neue TRANSPORTADRESSE mit den in *opt* angegebenen Werten zurück. Diese TRANSPORTADRESSE kann als Parameter *toaddr* an *t_conrq()* oder als Parameter *addr* an *t_getname()* übergeben werden.

-> opt

Zeiger auf eine Union, die die folgende in <cmx.h> definierte Struktur enthält:

```
struct t_optg5 {
    int t_optnr;                    /* Option Nr.              */
#define T_OPTG5 5
   struct t_addrpart t_nsap;        /* NSAP-Adresse */
   struct t_addrpart t_tsel;        /* T-Selektor */
   struct t_addrpart t_ssel;        /* S-Selektor */
   struct t_addrpart t_psel;        /* P-Selektor */
   /* Parameter fuer die TCP-TRANSPORTADRESSE */
   int portnumber;                  /* TCP-Portnummer          */
   /* Parameter fuer die BAM/HDLC-TRANSPORTADRESSE */
   unsigned char escaddr;           /* BAM/HDLC Escape-Adresse */
   unsigned char devtype;           /* Geraetetyp             */
   unsigned char proname[9];        /* BAM/HDLC-Prozessorname  */
}
```

-> t_optnr

Optionsnummer. Geben Sie an: T_OPTG5 in *t_optg5*.
Die folgende Datenstruktur ist in <cmx.h> definiert. Sie beschreibt die Einzelbestandteile einer TRANSPORTADRESSE in der Codierung, die durch die entsprechenden Protokolle und Standards, d. h. außerhalb von CMX, vorgegeben ist, und ohne die CMX-interne „Verpackung“.

```
struct t_addrpart {
    int t_maxlen;   /* Maximum lenght of t_bufp    */
    int t_type;     /* Type of NSAP/TSEL/SSEL/PSEL */
    int t_len;      /* Returned length of
                       Information in t_bufp       */
    char *t_bufp;   /* NSAP/TSAP/SSAP/PSAP         */
}
```

-> t_maxlen

Länge des vom Benutzerprogramm bereitgestellten Puffers *t_bufp*. Falls bei *t_getaddrpart* die in *t_bufp* zu speichernde Information länger als *t_maxlen* ist, wird ein Fehler angezeigt und keine Information in *t_bufp* geschrieben.

<- t_type (*t_getaddrpart()*)
-> t_type (*t_setaddrpart()*)

Dieser Parameter ist in der folgenden Tabelle beschrieben. Für die Strukturen *t_tsel*, *t_ssel* und *t_psel* gilt entweder der Wert T_VOID (existiert nicht) oder der Wert T_EXIST (existiert). Dabei ist die Kombination *t_type* = T_EXIST und *t_len* = 0 möglich. Bei den übrigen Werten sind jeweils die Codierung der network-service-spezifischen Adress-Information und die entsprechenden CMX-Adress-Formate angegeben. Bei *t_setaddrpart()* bedeutet *t_type* = T_VOID, dass der entsprechende

Bestandteil der TRANSPORTADRESSE unverändert bleibt. Bei *t_getaddrpart()* bedeutet *t_type* = T_VOID, dass der entsprechende Bestandteil der TRANSPORTADRESSE fehlt.

t_type	**z. Zt. benutzt in Adress-Format**	**Format**	**Bedeutung**
T_INETA	LANINET RFC1006	binär codiert (MSB) 4 oder 16 Oktette	IPv4- oder IPv6-Adresse
T_OSI	OSITYPE	OSI-Adresse wie in ISO 8348/ Add.2 angegeben	OSI-Adress-Format
T_NEA	WANNEA	binär codiert Byte [0] Prozessor Byte [1] Region#	NEA-Adress-Format
T_E164	WANSBKA WAN3SBKA	max. 15 Bytes ASCII codiert	ISDN-Nummer
T_E163	WANSBKA	max. 14 Bytes ASCII codiert	Tel.-Nummer
T_X121	WANSBKA WAN3SBKA	max. 15 Ziffern, mit 0xf aufgefüllt	X.25-Adresse
T_X21	WANSBKA	max. 20 Bytes ASCII codiert	X.21-Adresse
T_PVC	WANSBKA	binär codiert (MSB) 2 Oktette	
T_VOID			Adressteil existiert nicht bzw. bleibt unverändert
T_EXIST			Adressteil existiert (*t_len* = 0 ist möglich)

Tabelle 7: t_type

<- t_len (*t_getaddrpart()*)
-> t_len (*t_setaddrpart()*)

Länge des in *t_bufp* gespeicherten service-spezifischen Bestandteils der TRANSPORTADRESSE. Sie wird von *t_getaddrpart()* zurückgeliefert und an *t_setaddrpart()* übergeben.

<- t_bufp (*t_getaddrpart()*)
-> t_bufp (*t_setaddrpart()*)
: Zeiger auf einen vom Benutzerprogramm bereitgestellten Speicherbereich. *t_getaddrpart()* legt dort den entsprechenden service-spezifischen Bestandteil der TRANSPORTADRESSE ab. An *t_setaddrpart()* wird dort der neue Wert für den entsprechenden service-spezifischen Bestandteil der TRANSPORTADRESSE übergeben. Es handelt sich bei *t_nsap* um den Network Service, bei *t_tsel* um den Transport Service, bei *t_ssel* um den Session Service und bei *t_psel* um den Presentation Service.

t_nsap
: Diese Struktur beschreibt den network-service-spezifischen Bestandteil der TRANSPORTADRESSE.

t_tsel
: Diese Struktur beschreibt den transport-service-spezifischen Bestandteil der TRANSPORTADRESSE. Beim CMX-Adress-Format LANINET ist diese Struktur nicht relevant (*t_type* = T_VOID).

t_ssel
: Diese Struktur beschreibt den session-service-spezifischen Bestandteil der TRANSPORTADRESSE.

t_psel
: Diese Struktur beschreibt den presentation-service-spezifischen Bestandteil der TRANSPORTADRESSE.

<- portnumber (*t_getaddrpart()*)
-> portnumber (*t_setaddrpart()*)
: TCP-Portnummer. Ist nur relevant, wenn in der Struktur *t_nsap* das Strukturelement *t_type*= T_INET gesetzt ist.

escaddr
: BAM-Escape-Adresse. Nicht verwendet.

devtype
: BAM-Gerätetyp. Nicht verwendet.

proname
: BAM-Prozessorname. Nicht verwendet.

Hinweis

Der Wert *t_type* = T_INETA in *t_nsap* gilt sowohl für IPv4- als auch für IPv6-Adressen. Dagegen wird bei *t_getaddr()* mit *t_addrtype* = T_INETA in der Struktur *t_optg7* eine TRANSPORTADRESSEN-Schablone mit einer IPv4-Adresse angefordert und mit *t_addrtype* = T_INETA6 eine Schablone mit einer IPv6-Adresse.

Rückgabewerte

T_OK
: Erfolgreiche Beendigung. Information zur TRANSPORTADRESSE ist erfolgreich in die Datenstruktur geschrieben worden.

T_ERROR
: Fehler. Fragen Sie den Fehlercode durch Aufruf von *t_error()* ab.

Fehler

Tritt ein Fehler auf, so kann der Fehlercode durch Aufruf von *t_error()* abgefragt werden.

Mit Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können folgende Fehlerwerte auftreten:

T_WPARAMETER
: Der in *addr* oder *opt* angegebene Wert ist ein NULL-Zeiger oder *addr* zeigt nicht auf eine Partneradresse.

Verwendung

Anwendungen können das Ergebnis von *t_getaddr()* und *t_conin()* zur eigenen Nutzung ändern.

8.6.18 t_getloc - LOKALEN NAMEN abfragen

t_getloc liefert entweder den LOKALEN NAMEN des Objekts *globname* aus dem TS-Directory 1 (*globname* !=NULL) oder ein Template (*globname* = NULL) zurück.

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

-> globname !=NULL

gibt den GLOBALEN NAMEN des Objekts im TS-Directory an und wird in der Form NP5.NP4.NP3.NP2.NP1 als String (NULL-terminated) erwartet. Die Elemente NPi repräsentieren die Namensteile des GLOBALEN NAMENS in aufsteigender hierarchischer Ordnung von links nach rechts. Hat ein NPi keinen Wert, so muss der Separator '.' angegeben werden, wenn dieser Namensteil von mindestens einem weiteren Namensteil gefolgt wird, der in der Hierarchie höher ist (mehrere '.' am Ende können weggelassen werden). Mindestens einer der Namensteile NPi muss einen Wert haben. '.' als Bestandteil eines Namensteils muss mit '\.' entwertet werden.

-> globname=NULL

bewirkt, dass *t_getloc* einen Zeiger auf ein Template mit der folgenden Datenstruktur zurückgibt:

```
mn_header | rest .........
01000032    000e0000 00000100 00043131 31310000 00000000 /*LANINET TSEL*/
                      0040 000a6161 61616161 61613300 /*EMSNA   TSEL*/
                      241f 00086161 61616161 61610000
                      |
                      CX_WANNEA + CX_LANSBKA + CX_RFC1006 +
                      CX_WANSBKA + WAN3SBKA + LOOPSBKA + OSITYPE TSEL
```

Um die Werte des Template zu verändern, verwenden Sie die Funktion *t_setlocpart*.

-> opt

Zeiger auf eine Union mit Systemoption oder NULL.

CMX in UNIX ignoriert diesen Parameter. Die folgende Struktur ist in <cmx.h> nur aus Gründen der Kompatibilität zum CMX in BS2000/OSD definiert:

```
    struct t_optg1 {
->      int t_optnr;          /*Optionsnr.                          */
->      int t_evref;          /*Referenzpunkt für Systemereignis */
->      char *t_buf[200];     /* Arbeitsbereich                     */
    }
```

t_optnr
: Optionsnummer. Geben Sie an: T_OPTG1 in *t_optg1*

t_evref
: Dieses Feld wird von BS2000/OSD benutzt und wird von CMX in UNIX nicht unterstützt.

t_buf [200]
: Dieses Feld wird von BS2000/OSD benutzt und wird von CMX in UNIX nicht unterstützt.

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_getloc()* einen Zeiger auf einen statischen Bereich mit dem LOKALEN NAMEN zurück. Im Fehlerfall liefert *t_getloc()* den NULL-Zeiger zurück.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden. Es können Fehlerwerte vom Fehlertyp T_DSTEMP_ERR, T_DSCALL_ERR, T_DSPERM_ERR und T_DSWARNING auftreten.

Mit Fehlerklasse T_DSPAR_ERR können die folgenden Fehlerwerte auftreten:

T_DIRERR
: Das TS-Directory DIR1 ist nicht vorhanden.

T_NAMERR
: Der in *globname* angegebene GLOBALE NAME existiert im TS-Directory DIR1 nicht.

T_ILLNAM
: Der in *globname* angegebene GLOBALE NAME ist syntaktisch falsch (zu viele Namensteile, ungültige Längen der Namensteile, ungültige Zeichen innerhalb des Namens).

T_PROPER
: Dem in *globname* angegebenen GLOBALEN NAMEN ist kein LOKALER NAME zugeordnet.

Mögliche Fehlerwerte mit Fehlerklasse T_DSSYS_ERR sind die in <errno.h> definierten Systemfehlermeldungen.

Mit Fehlerklasse T_DSILL_VERS kann folgender Fehlerwert auftreten:

T_NOTSPEC
: Die im Prozess eingebundene Version der CMX-Bibliothek und die CMX-Ablaufumgebung sind unverträglich.

Mit Fehlerklasse T_DSINT_ERR sind folgende Fehlerwerte möglich:

T_TIMOUT
: Der TNS-Dämon tnsxd (CMX_1) antwortet nicht innerhalb eines Zeitraums von 20 Sekunden.

T_PROT
: Im tnsxd (CMX_1)-Protokoll treten Fehler auf.

T_LFILE
: Das TS_Directory DIR1 hat kein korrektes Format.

Verwendung

Der oben genannte statische Bereich wird bei jedem Aufruf überschrieben. Der Aufrufende muss den Bereich kopieren, wenn er gesichert werden soll. Der Menge der zu kopierenden Daten kann aufgrund der Größe des Feldes *t_mnlng* in der Struktur *t_myname* bestimmt werden.

8.6.19 t_getlocpart, t_setlocpart - Lesen oder Ändern von Adress-Information in LOKALER NAME

t_getlocpart - Adress-Information aus LOKALEM NAMEN lesen
t_setlocpart - Adress-Information in LOKALEM NAMEN ändern

t_getlocpart() liefert zu dem in *addr* angegebenen LOKALEN NAMEN die einzelnen service-spezifischen Adress-Informationen in *opt* zurück.

t_setlocpart() liefert in *newaddr* einen geänderten LOKALEN NAMEN, wobei das aufrufende Program in *addr* den ursprünglichen LOKALEN NAMEN und in *opt* die Modifikation der einzelnen service-spezifischen Adress-Information angibt. Bei der transport-service-spezifischen Adress-Information gibt es folgende Besonderheiten:
Statt *t_tsel* wird beim CMX-Adress-Format LANINET *portnumber* und beim CMX-Adress-Format EMSNA *lu_size*, *luname* und *lunumber* verwendet. Falls in einem LOKALEN NAMEN von den übrigen CMX-Adress-Formaten mehrere gleichzeitg gültig sind, wird vorausgesetzt, dass bei allen der T-Selektor denselben Wert und dieselbe Codierung besitzt.

```
#include <cmx.h>
int   t_getlocpart (const union t_address *addr,
                    t_optg *opt);
int   t_setlocpart (const union t_address *addr,
                    union t_address *newaddr,
                    const t_optg *opt);
```

-> addr

Zeiger auf eine Union *t_address*, die den LOKALEN NAMEN einer TS-Anwendung enthält. Das CMX-Programm erhält den LOKALEN NAMEN zum Beispiel durch Aufruf von *t_getloc()*.

<- newaddr

Zeiger auf eine Union *t_address*. Hier gibt *t_setlocpart()* den geänderten LOKALEN NAMEN zurück. Dieser kann als Parameter *t_myname* bei *t_attach()* oder als Parameter *fromaddr* bei *t_conrq()* verwendet werden.

-> opt

Zeiger auf eine Union, die eine der folgenden, in <cmx.h> definierten Strukturen enthält:

```
struct t_optg5 {
    int t_optnr;                    /* Option Nr.                */
#define T_OPTG5 5
    struct t_addrpart  t_nsap;      /* Information zum NSAP      */
    struct t_addrpart  t_tsel;      /* Information zum TSEL      */
    struct t_addrpart  t_ssel;      /* Information zum SSEL      */
    struct t_addrpart  t_psel;      /* Information zum PSEL      */

    /* Parameter für TCP TRANSPORTADRESSE */
    int portnumber;                 /* TCP-Portnummer            */

    /* Parameter für BAM/HDLC TRANSPORTADRESSE */
    unsigned char escaddr;          /* BAM/HDLC Escape-Adresse   */
    unsigned char devtype;          /* Gerätetyp                 */
    unsigned char proname[9];       /* BAM/HDLC Prozessorname    */
}


struct t_optg6 {
    int t_optnr;                    /* Option Nr.                */
#define T_OPTG6 6
    struct t_addrpart  t_nsap;      /* Information zum NSAP      */
    struct t_addrpart  t_tsel;      /* Information zum TSEL      */
    struct t_addrpart  t_ssel;      /* Information zum SSEL      */
    struct t_addrpart  t_psel;      /* Information zum PSEL      */

    /* Parameter für TCP TRANSPORTADRESSE */
    int portnumber;                 /* TCP-Portnummer            */

    /* Parameter für SNA LUNAME */
    int lu_size;                     /* Länge von LUNAME         */

    /* Parameter für BAM/HDLC TRANSPORTADRESSE */
```

```
        unsigned char escaddr;          /* BAM/HDLC Escape-Adresse  */
        unsigned char devtype;          /* Gerätetyp                */
        unsigned char proname[9];       /* BAM/HDLC Prozessorname   */

        /* Parameter für SNA LUNAME */
        unsigned char luname[8];        /* LU-Name (max. 8) +       */
        unsigned char lunumber;         /* LU-Nummer                */
    }
```

-> t_optnr

Nummer der Option. Geben Sie an: T_OPTG5 wenn Sie *t_optg5* verwenden, oder T_OPTG6 im Falle von *t_optg6*. *t_optg6* sollte nur mit dem Adress-Format CX_EMSNA verwendet werden, bei dem der LU-Name Teil des LOKALEN NAMENS ist.

Die folgende Datenstruktur ist in <cmx.h> definiert. Sie beschreibt die Einzelbestandteile eines LOKALEN NAMENS in der Codierung, die durch die entsprechenden Protokolle und Standards, d. h. außerhalb von CMX, vorgegeben ist, und ohne die CMX-interne „Verpackung“.

```
struct t_addrpart {
     int t_maxlen;              /* Länge des Puffers          */
     int t_type;                /* T_EXIST oder T_VOID        */
     int t_len;                 /* Länge der Information      */
     char *t_bufp;              /* Puffer                     */
  }
```

-> t_maxlen

Länge des vom Benutzerprogramm zur bereitgestellten Puffers *t_bufp*. Falls bei *t_getlocpart* die in *t_bufp* zu speichernde Information länger als *t_maxlen* ist, wird ein Fehler angezeigt und keine Information in *t_bufp* geschrieben.

<- t_type (*t_getlocpart()*)
-> t_type (*t_setlocpart()*)

Der Wert T_VOID bedeutet bei *t_getlocpart()*, dass die entsprechende Adress-Information nicht vorhanden ist, und bei *t_setlocpart()*, dass die entsprechende Adress-Information unverändert bleiben soll.
Der Wert T_EXIST bedeutet bei *t_getlocpart()*, dass die entsprechende Adress-Information vorhanden ist, und bei *t_setlocpart()*, dass die entsprechende Adress-Information durch den angegebenen neuen Wert ersetzt werden soll.

<- t_len (*t_getlocpart()*)
-> t_len (*t_setlocpart()*)

Länge der im Puffer *t_bufp* abgelegten service-spezifischen Adress-Information.

<- t_bufp (*t_getlocpart()*)
-> t_bufp (*t_setlocpart()*)
: Zeiger auf einen vom Benutzerprogramm bereitgestellten Speicherbereich.
t_getlocpart() legt dort die entsprechende service-spezifische Adress-Information des LOKALEN NAMENS ab.
An *t_setlocpart()* wird dort der neue Wert für die entsprechende service-spezifische Adress-Information des LOKALEN NAMENS übergeben.

t_tsel
: Diese Struktur beschreibt die transport-service-spezifische Adress-Information des LOKALEN NAMENS (T-Selektor).

t_ssel
: Diese Struktur beschreibt die session-service-spezifische Adress-Information des LOKALEN NAMENS (S-Selektor).

t_psel
: Diese Struktur beschreibt die presentation-service-spezifische Adress-Information des LOKALEN NAMENS (P-Selektor).

<- portnumber (*t_getlocpart()*)
-> portnumber (*t_setlocpart()*)
: TCP-Portnummer (transport-service-spezifische Adress-Information für das CMX-Adress-Format LANINET).

<- lu_size (*t_getlocpart()*)
-> lu_size (*t_setlocpart()*)
: Länge des in *luname* angebenen LU-Namens (transport-service-spezifsche Adress-Information für das CMX-Adress-Format EMSNA).

escaddr
: BAM Escape-Adresse. Dieser Parameter wird nicht verwendet.

devtype
: BAM-Gerätetyp. Dieser Parameter wird nicht verwendet.

proname
: BAM-Prozessorname. Dieser Parameter wird nicht verwendet.

<- luname (*t_getlocpart()*)
-> luname (*t_setlocpart()*)
: SNA-LU-Name (transport-service-spezifische Adress-Information für das CMX-Adress-Format EMSNA).

<- lunumber (*t_getlocpart()*)
-> lunumber (*t_setlocpart()*)
SNA-LU-Nummer (transport-service-spezifische Adress-Information für das CMX-Adress-Format EMSNA).

Rückgabewerte

T_OK
Erfolgreich beendet.

T_ERROR
Fehler. Rufen Sie *t_error()* auf, um den Fehlercode abzufragen.

Fehler

Tritt ein Fehler auf, so kann der Fehlercode durch Aufruf von *t_error()* abgefragt werden.

Mit Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können folgende Fehlerwerte auftreten:

T_WPARAMETER
Der in *addr* oder *opt* angegebene Wert ist ein NULL-Zeiger oder *addr* zeigt nicht auf eine Union *t_address*.

8.6.20 t_getname - GLOBALEN NAMEN abfragen (get name)

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

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

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

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

Der GLOBALE NAME wird von CMX als mit NIL endender String in der Form NP5.NP4.NP3.NP2.NP1 geliefert. Die NPi (i=1,2,3,4,5) sind die Namensteile des GLOBALEN NAMENS. Dabei ist NP5 der Namensteil[5], also der Namensteil der unteren Hierarchiestufe. NP1 ist der Namensteil[1], also der in der Hierarchie höchste Namensteil. Die restlichen Namensteile sind in von links nach rechts aufsteigender hierarchischer Reihenfolge angegeben.

Ist bei einem GLOBALEN NAMEN einer der Namensteile nicht belegt (z. B. NP4) und folgt diesem Namensteil noch ein Namensteil höherer Hierarchie (z. B. NP3), so wird von dem nicht belegten Namensteil das Trennzeichen (.) dennoch geliefert.

Eine Folge von Trennzeichen am Ende des Wertes von *globname* wird weggelassen.

Der GLOBALE NAME wird dann von CMX wie folgt angegeben: „NP5..NP3“

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

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

-> addr

Zeiger auf einen Bereich mit der TRANSPORTADDRESSE

-> opt

Zeiger auf eine Einheit mit Systemoptionen oder NULL.

CMX in Solaris ignoriert diesen Parameter. Die folgende Struktur ist in <*cmx.h*> nur aus Kompatibilität zum CMX in BS2000/OSD definiert.

```
    struct  t_optg1 {
->     int  t_optnr;          /* Options-Nr.          */
->     int  t_evref;          /* System Ereignis
                                 Referenzpunkt        */
->     char *t_buf[200];      /* Arbeitsbereich       */
    };
```

t_optnr

Optionsnummer. Anzugeben ist:
T_OPTG1 in *t_optg1*

t_evref

Dieses Feld wird von BS2000/OSD benutzt und wird von CMX in Solaris nicht unterstützt.

t_buf[200]

Dieses Feld wird von BS2000/OSD benutzt und wird von CMX in Solaris nicht unterstützt.

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_getname()* den Zeiger auf einen Bereich mit dem GLOBALEN NAMEN zurück.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Es können Fehlerwerte vom Fehlertyp T_DSTEMP_ERR, T_DSCALL_ERR,T_DSPERM_ERR und T_DSWARNING auftreten.

Mit Fehlerklasse T_DSPAR_ERR können folgende Fehlerwerte auftreten:

Das TS-Directory DIR1 ist nicht vorhanden.

T_LENERR
: Das Längenfeld *t_palng*, das in der in *addr* angegebenen TRANSPORT-ADRESSE enthalten ist, hat einen ungültigen Wert.

Mögliche Fehlerwerte mit Fehlerklasse T_DSSYS_ERR sind die in <*errno.h*> definierten Systemfehlermeldungen.

Mit Fehlerklasse T_DSILL_VERS kann folgender Fehlerwert auftreten:

T_NOTSPEC
: Die im Prozess eingebundene CMX-Version und die CMX-Ablaufumgebung sind unverträglich.

Mit Fehlerklasse T_DSINT_ERR sind folgende Fehlerwerte möglich:

T_TIMOUT
: Der TNS-Daemon *tnsxd* antwortet nicht innerhalb eines Zeitraums von 20 Sekunden.

T_PROT
: Im Protokoll mit *tnsxd* treten Fehler auf.

T_LFILE
: Das TS-Directory 1 (DIR1) hat kein korrektes Format.

Mit Fehlerklasse T_DSMESSAGE ist folgender Fehlerwert möglich:

T_LEAFNO
: Im TS-Directory 1 existiert entweder kein oder mehr als ein GLOBALER NAME, dem die in *addr* angegebene TRANSPORTADRESSE zugeordnet ist.

Siehe auch

t_error(), TNS in CMX „Betrieb und Administration“ [1] oder [2]

8.6.21 t_info - Informationen über CMX abfragen (information)

t_info() informiert über die maximale TIDU-Länge. Die Information steht in der Regel erst zur Verfügung, nachdem die Transportverbindung vollständig aufgebaut worden ist.

```
#include <cmx.h>
int   t_info (const int *tref,
              t_opti *opt);
```

-> tref

Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

<> opt

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

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

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

   struct t_opti2 {
->     int t_optnr;           /* Options-Nr.          */
<-     int t_evref;           /* System Referenz
                                 Ereignispunkt        */
<-     int t_buffer[180];     /* Puffer für Name
                                 Service Ausgabe      /*
    };
```

t_optnr

Optionsnummer. Anzugeben ist:
T_OPTI1 bei *t_opti1*
T_OPTI2 bei *t_opti2*

t_maxl
: In dieses Feld trägt CMX die maximale Länge der TIDU ein.

  Der Wert gibt an, wieviele Bytes beim Datentransfer über diese Verbindung maximal pro Aufruf an CMX übergeben bzw. von CMX empfangen werden können.

t_evref
: Dieses Feld wird von BS2000 Anwendungen verwendet und wird von CMX in Solaris nicht unterstützt.

t_buffer[180]
: Dieses Feld wird von BS2000 Anwendungen verwendet und wird von CMX in Solaris nicht unterstützt.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Der Zeiger *opt* (!= NULL) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder die gewünschte Information steht noch nicht zur Verfügung.

  t_info() kann erst ausgeführt werden, wenn die Verbindung etabliert ist, da dann erst Angaben über die Verbindung gemacht werden können.

T_WPARAMETER
: Die durch *opt* übergebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

8.6.22 t_perror - CMX-Fehlermeldung decodiert ausgeben

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

Im Parameter *s* kann ein zusätzlicher erklärender Text angegeben werden, z.B Angabe auf welchen CMX-Aufruf und welche TS-Anwendung sich der Fehler bezieht. Die Ausgabe von *t_perror()* ist folgendermaßen aufgebaut:

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

Der begleitende Text wird aus der Meldungsdatei *cmxlib.cat* übernommen. Er wird nicht ausgegeben, wenn *cmxlib.cat* an Ihrem System nicht vorhanden ist. Das Format von *cmxlib.cat* ist abhängig vom Betriebssystem und der gesetzten Sprachvariablen. Nähere Informationen dazu entnehmen Sie bitte dem jeweiligen Systemhandbuch.

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

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

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

Dateien

cmxlib.cat - Meldungsdatei

Siehe auch

t_error(), t_strerror()

8.6.23 t_preason - Verbindungsabbaugründe decodieren und ausgeben

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

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

Aufbau der Ausgabe von *t_preason()*:

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

Der begleitende Text wird aus der Meldungsdatei *cmxlib.cat* übernommen. Er wird nicht ausgegeben, wenn *cmxlib.cat* an Ihrem System nicht vorhanden ist. Das Format von *cmxlib.cat* ist abhängig vom Betriebssystem und der gesetzten Sprachvariablen. Nähere Informationen dazu entnehmen Sie bitte dem jeweiligen Systemhandbuch.

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

-> s

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

-> reason

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

Dateien

cmxlib.cat - Meldungsdatei

Siehe auch

t_disin(), t_strreason()

8.6.24 t_redin - Umgelenkte Verbindung annehmen (redirection indication)

t_redin() nimmt ein zuvor mit *t_event()* gemeldetes Ereignis T_REDIN entgegen. T_REDIN zeigt an, dass 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 ursprünglichen Prozess zurückgegeben oder mit *t_disrq()* abgebaut werden.

Der Aufruf *t_redin()* liefert:

– die Prozess-Identifikation des rufenden Prozesses,

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

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

```
#include <cmx.h>
int   t_redin (const int *tref,
               int *pid,
               t_opt23 *opt);
```

t_opt23 *opt;
: Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

<- pid
: Zeiger zu einem Feld, in dem CMX die Prozess-Identifikation des Prozesses zurückliefert, der die Verbindung umlenkt.

<> opt

Für *opt* ist der NULL-Zeiger oder der Zeiger zu einer Union mit Systemoptionen anzugeben.

Damit können Benutzerdaten entgegengenommen werden, die der rufende Prozess bei der Anforderung der Verbindungsumlenkung (*t_redrq()*) mitgegeben hat.

Wird *opt* = NULL angegeben, so vernichtet CMX die Benutzerdaten. Hat der rufende Prozess keine Benutzerdaten angegeben, so liefert CMX die angegebenen Standardwerte.

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

```
    struct  t_optc2 {
->     int  t_optnr;      /* Options-Nr. */
<-     char *t_udatap;    /* Datenpuffer */
< >    int  t_udatal;     /* Länge des Datenpuffers */
    };
    struct t_optc3 {
->     int  t_optnr;      /* Options-Nr. */
<-     char *t_udatap;    /* Datenpuffer */
< >    int   _udatal;     /* Länge des Datenpuffers */
<-     int  t_xdata;      /* Vorrangdaten-Auswahl */
<-     int  t_timeout;    /* Inaktiv-Zeit */
->     int  t_ucepid;     /* Benutzerreferenz der
                             Verbindung */
   };
    struct t_optc4 {
->     int  t_optnr;      /* Options-Nr. */
<-     char *t_udatap;    /* Datenpuffer */
< >    int  t_udatal;     /* Länge des Datenpuffer */
<-     int  t_xdata;      /* Vorrangdaten-Auswahl */
<-     int  t_timeout;    /* Inaktiv-Zeit */
->     int  t_ucepid;     /* Benutzerreferenz der
                             Verbindung */
<->    int  t_tid_valid; /* T_YES / T_NO */
<-     void *t_tid;       /* Pointer auf Thread-ID */
};
```

t_optnr

Optionsnummer. Anzugeben ist:
T_OPTC2 bei *t_optc2*
T_OPTC3 bei *t_optc3*
T_OPTC4 bei *t_optc4* für Multithreading

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

  Standardwert bei Angabe von *opt* = NULL: undefiniert

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

  Standardwert bei Angabe von *opt* = NULL: 0.

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

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

t_ucepid
: In diesem Feld kann eine beliebige Benutzerreferenz dieser Verbindung an CMX übergeben werden.

  Diese Benutzerreferenz kann dem laufenden Prozess als Option im Folgenden beim *t_event()* von CMX zurückgeliefert werden.

  Damit kann der laufende Prozess, falls er mehrere Verbindungen hält, Ereignisse der entsprechenden Verbindung über ein selbst gewähltes Merkmal zuordnen. Die Benutzerreferenz ist eine Alternative zu der von CMX gewählten Transportreferenz *tref*.

  Standardwert bei Angabe von *opt* = NULL: 0

t_tid_valid
: Kennzeichnung, ob Feld *t_tid* gültig ist; vor dem Aufruf muss hier der Wert T_YES oder T_NO eingetragen sein. Bei T_YES erwartet die Applikation von CMX in **t_tid* die Thread-ID des Threads, der die Verbindung umlenkt. In diesem Fall muss die Applikation in **t_tid* einen Bereich der Größe *pthread_t* bereitstellen.

  Nach dem Aufruf liefert CMX in **t_tid* die ID des Threads zurück, der die Verbindung umgelenkt hat und zeigt dies im Feld *t_tid_valid* mit T_YES an; ist die ID nicht verfügbar, wird T_NO gesetzt.

t_tid

Zeiger zu einem Feld vom Typ *pthread_t*, in dem CMX die Thread-ID des Threads zurückliefert, der die Verbindung umlenkt; *t_tid* ist nur dann gültig, wenn *t_tid_valid* den Wert T_YES enthält. Das Feld wird ignoriert, wenn vor dem Aufruf in *t_tid_valid* T_NO übergeben wird.

Siehe auch den Hinweis bei *t_redrq()* zur Verbindungsumlenkung bei Multithreading auf Seite 197.

Rückgabewert

T_OK
Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
Mindestens einer der Zeiger *opt* (!= NULL) oder *t_udatap* (!= NULL und *t_ndatal* != 0) zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
Der Prozess ist in keiner TS-Anwendung angemeldet oder für die in *tref* angegebene Verbindung ist kein T_REDIN angezeigt.

T_WPARAMETER
Die in *opt* angegebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.25 t_redrq - Verbindung umlenken (redirection request)

t_redrq() lenkt die angegebene Verbindung an einen anderen Prozess um. Der empfangende Prozess wird durch den Parameter *pid* festgelegt. Er muss in der TS-Anwendung angemeldet sein, zu der die umzulenkende Verbindung gehört.

Beim *t_redrq()* kann der laufende Prozess in den Optionen *opt*:

– eine Wartezeit vereinbaren; während dieser Zeit wird auf die Anmeldung des empfangenden Prozesses in der TS-Anwendung gewartet,

– Benutzerdaten angeben, die dem empfangenden Prozess bei der Übernahme der Verbindung übergeben werden. Über die Benutzerdaten kann man dem empfangenden Prozess z. B. mitteilen, zu welcher TS-Anwendung die Verbindung gehört.

Die Verbindung ist nach dem *t_redrq()* 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 TIDU dieser Verbindung stückweise mit *t_datain()* (Rückgabewert: n > 0) abgeholt wird.

```
#include <cmx.h>
int   t_redrq (const int *tref,
               const int *pid,
               const t_opt12 *opt);
```

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

-> pid
: Zeiger zu einem Feld, in dem die Prozess-Identifikation des gerufenen Prozesses anzugeben ist.

-> opt
: Für *opt* ist der Wert NULL oder der Zeiger zu einer Union mit Benutzeroptionen anzugeben.

Damit kann man dem gerufenen Prozess bei der Verbindungsumlenkung Informationen mitschicken. Der gerufene Prozess nimmt diese mit der Verbindungsumlenkung entgegen. Wird *opt* = NULL angegeben, so stellt CMX dem gerufenen Prozess die angegebenen Standardwerte zu.

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

```
    struct  t_optc1 {
->     int  t_optnr;      /* Options-Nr. */
->     char *t_udatap;    /* Datenpuffer */
->     int  t_udatal;     /* Länge des Datenpuffers */
       int  t_xdata;      /* Vorrangdaten-Auswahl */
->     int  t_timeout;    /* Wartezeit auf Anmeldung */
    };
    struct  t_optc2 {
->     int  t_optnr;      /* Options-Nr. */
->     char *t_udatap;    /* Datenpuffer */
->     int  t_udatal;     /* Länge des Datenpuffers */
    };
    struct  t_optc4 {
->     int  t_optnr;      /* Options-Nr. */
->     char *t_udatap;    /* Datenpuffer */
->     int  t_udatal;     /* Länge des Datenpuffers */
<-     int  t_xdata;      /* Vorrangdaten-Auswahl */
->     int  t_timeout;    /* Wartezeit auf Anmeldung */
->     int  t_ucepid;     /* Benutzerreferenz der
                             Verbindung */
->     int  t_tid_valid; /* T_YES / T_NO */
<-     void *t_tid;       /* Pointer auf Thread-ID */
};
```

t_optnr
: Optionsnummer. Anzugeben ist:
T_OPTC1 bei *t_optc1*
T_OPTC2 bei *t_optc2*
T_OPTC4 bei *t_optc4* für Multithreading

t_udatap
: Zeiger zu einem Bereich mit Benutzerdaten, die dem empfangenden Prozess zugestellt werden sollen.

  Standardwert bei Angabe von *opt* = NULL: undefiniert

t_udatal
: Anzahl der aus dem Datenbereich *t_udatap* zu übertragenden Byte. Die maximal mögliche Anzahl ist in <*cmx.h*> als T_RED_SIZE definiert.

Wird *t_udatal* = 0 angegeben, so wird *t_udatap* nicht ausgewertet.

Der Maximalwert für *t_udatal* ist abhängig vom Transportsystem (siehe Freigabemitteilung).

Standardwert bei Angabe von *opt* = NULL: 0

t_xdata

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

t_timeout

Für den Parameter kann eine Wartezeit in Sekunden angegeben werden. Während dieser Zeit wartet der laufende Prozess synchron auf die Anmeldung des empfangenden Prozesses in derselben TS-Anwendung.

Der Wartezustand wird durch die erwartete Anmeldung beendet oder durch Signale abgebrochen.

Ist die Wartezeit abgelaufen, ohne dass sich der empfangende Prozess für die entsprechende TS-Anwendung angemeldet hat, oder wird der Wartezustand durch Signale abgebrochen, so endet der Aufruf mit einer Fehlermeldung.

Mögliche Angaben für *t_timeout*:

T_NOLIMIT

Es wird keine begrenzte Wartezeit vorgegeben. Der Prozess wartet unbegrenzt auf die erwartete Anmeldung des empfangenden Prozesses.

T_NO

Der Prozess wartet nicht. Er wird sofort fortgesetzt. Liegt die erwartete Anmeldung nicht vor, so endet der Aufruf mit einer Fehlermeldung.

n > 0

Der laufende Prozess wartet n Sekunden auf die Anmeldung. Er wird in dieser Zeit suspendiert. Erfolgt die Anmeldung in dieser Zeitspanne nicht, so endet der Aufruf mit einer Fehlermeldung.

Standardwert bei Angabe von *opt* = NULL: T_NO

t_tid_valid

t_tid_valid gibt an, ob in **t_tid* eine Thread-ID übergeben wird. Bei T_YES enthält **t_tid* die Thread-ID des Threads, an den die Verbindung umgelenkt werden soll. Bei T_NO wird Feld *t_tid* nicht ausgewertet.

t_tid

Zeiger zu einem Feldvom Typ *pthread_t* mit der Thread-ID des Threads, zu dem die Verbindung umgelenkt werden soll. Dieser Thread muss sich im Prozess **pid* befinden.

t_tid ist nur dann gültig, wenn *t_tid_valid* den Wert T_YES enthält. Das Feld wird ignoriert, wenn vor dem Aufruf in *t_tid_valid* T_NO übergeben wird.

Hinweis zur Verbindungsumlenkung bei Multithreading

Die beiden Funktionen *t_redrq()* und *t_redin()* verwenden die Prozess-ID, um den Prozess zu identifizieren, zu dem eine Verbindung umgelenkt werden soll bzw. von welchem Prozess die Verbindung erhalten wurde.

Die Verbindungsumlenkung ist sowohl zu einem Thread des eigenen als auch zu einem Thread eines fremden Prozesses möglich.

Nachstehende Tabelle gibt Auskunft darüber, wie die Umlenkung erfolgt:

Caller-PID	Receiver-PID	Thread-ID	Umlenkung erfolgt
p1	p2	–	zum ersten verfügbaren Thread im Prozess p2
p1	p2	k	zu Thread k im Prozess p2 (*)
p1	p1	–	zum ersten verfügbaren Thread im eigenen Prozess
p1	p1	k	zu Thread k im eigenen Prozess, sofern k nicht die eigene Thread-ID ist

Tabelle 8: Verbindungsumlenkung bei Multithreading

Erläuterungen:

(*) der Aufrufer müsste allerdings die Thread-ID im empfangenden Prozess ermitteln, beispielsweise durch Interprozesskommunikation.

"verfügbar" heißt, dass die Empfangskriterien erfüllt sind (*t_conlim* noch nicht erschöpft, T_REDIRECT für *t_apmode* gesetzt und mit demselben LOKALEN NAMEN angemeldet).

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Mindestens einer der Zeiger *opt* (!= NULL) oder *t_udatap* (!= NULL und *t_ndatal* != 0) zeigt nicht in den Prozess-Adressraum.

T_ETIMEOUT
: Die Wartezeit des laufenden Prozesses, während der er auf die Anmeldung des empfangenden Prozesses in derselben TS-Anwendung wartet, ist abgelaufen.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet, oder die in *tref* angegebene Verbindung ist nicht vorhanden, oder der Datenfluss auf der Verbindung *tref* ist sendeseitig gesperrt, oder eine TIDU oder eine ETSDU ist noch nicht vollständig gelesen.

T_WPARAMETER
: Der beim Parameter *pid* angegebene Prozess ist der laufende Prozess, ode der in *pid* angegebene Prozess ist in dieser TS-Anwendung nicht angemeldet, oder der in *pid* angegebene Prozess hat bei der Anmeldung mit *t_attach()* in *t_apmode* nicht T_REDIRECT gesetzt, oder die in *opt* angegebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte.

T_WCONN_LIMIT
: Der in *pid* angegebene Prozess hat alle Verbindungen bereits ausgeschöpft, die ihm verfügbar sind.

T_WRED_LIMIT
: Der Grenzwert gleichzeitiger möglicher Verbindungsumlenkungen ist überschritten.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.26 t_setaddrpart - Information zur TRANSPORTADRESSE einfügen

Siehe Abschnitt „t_getaddrpart, t_setaddrpart - Lesen oder Ändern der Adress-Information in TRANSPORTADRESSE“ auf Seite 171.

8.6.27 t_setlocpart - TSEL aus lokalen Namen übergeben

Siehe Abschnitt „t_getlocpart, t_setlocpart - Lesen oder Ändern von Adress-Information in LOKALER NAME“ auf Seite 178.

8.6.28 t_setopt - Optionen in CMX schalten (set options)

Mit *t_setopt* können Optionen in CMX an- und ausgeschaltet werden.

In dieser Version wird nur die Option T_DEBUG zum Schalten des Bibliotheksverfolgers angeboten.

```
#include <cmx.h>
int   t_setopt (int component,
                const t_opts *opt);
```

-> component

Gibt an, in welcher CMX-Komponente die Option gesetzt werden soll.

Mögliche Werte:

T_LIB

Bei der gesetzten Option handelt es sich um eine Bibliotheksoption.

-> opt

Zeiger auf eine Union, die eine Optionsstruktur enthält.

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

```
    struct  t_opts1 {
->     int  t_optnr;      /* Options-Nr.  */
->     int  t_optname;    /* Options-Name */
```

```
->      char *t_optvalue;  /* Zeiger auf Optionen String */
    };
```

t_optnr
: Optionsnummer. Anzugeben ist T_OPTS1.

t_optname
: Gibt die Option an, die an- oder ausgeschaltet werden soll.

  Mögliche Werte:

  T_DEBUG
  : Verfolger schalten.

t_optvalue
: Zeiger zu einem String (abgeschlossen mit NIL), der den Optionswert enthält. Ist der String leer, wird die Option aus *t_optname* ausgeschaltet. Der Inhalt des Strings hängt vom Wert von *t_optname* ab.

  t_optname = T_DEBUG: Das Format der Verfolgeroptionen ist dasselbe wie bei der Umgebungsvariablen CMXTRACE (siehe CMX „Betrieb und Administration“ [1] oder [2]).

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_WPARAMETER
: Der in *component* angegebene Wert ist ungültig oder die in *opt* angegebene Option hat ein falsches Format oder enthält falsche Werte.

8.6.29 t_strerror - CMX-Fehlermeldung decodieren

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

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

Der Klartext setzt sich zusammen aus Fehlersymbolen, wie in <*cmx.h*> definiert, und begleitendem Text. Jedes Fehlersymbol wird von \t eingeleitet. Jeder begleitende Text wird mit \n abgeschlossen.

Der begleitende Text wird aus der Meldungsdatei *cmxlib.cat* übernommen. Er wird nicht ausgegeben, wenn *cmxlib.cat* an Ihrem System nicht vorhanden ist. Das Format von *cmxlib.cat* ist abhängig vom Betriebssystem und der gesetzten Sprachvariablen. Nähere Informationen dazu entnehmen Sie bitte dem jeweiligen Systemhandbuch.

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

-> code

Für *code* ist die Darstellung der Fehlermeldung anzugeben, die dem Prozess beim Aufruf von *t_error()* von CMX übergeben wurde.

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_strerror()* den Zeiger auf einen Bereich mit der Klartextdarstellung der CMX-Fehlermeldung als String in C-Notation zurück.

Wird in *code* ein nicht definierter Wert angegeben, so liefert *t_strerror()* den Klartext:

```
"\t<code> Nicht decodierbar\n"
```

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

Dateien

cmxlib.cat - Meldungsdatei

Siehe auch

t_error(), t_perror()

8.6.30 t_strreason - Verbindungsabbaugründe decodieren

t_strreason() decodiert Verbindungsabbaugründe, die dem Prozess beim Aufruf von *t_disin()* in sedezimaler Darstellung übergeben werden. *t_strreason()* liefert einen Zeiger auf einen statischen Bereich, der die Klartextdarstellung zu dem in *reason* übergebenen Grund für den Verbindungsabbau enthält. Der Klartext setzt sich zusammen aus dem Symbol für den Verbindungsabbaugrund, wie in <*cmx.h*> definiert, und einem begleitenden Text. Das Symbol für den Verbindungsabbau wird von \t eingeleitet. Der begleitende Text wird mit \n abgeschlossen.

Der begleitende Text wird aus der Meldungsdatei *cmxlib.cat* übernommen. Er wird nicht ausgegeben, wenn *cmxlib.cat* an Ihrem System nicht vorhanden ist. Das Format von *cmxlib.cat* ist abhängig vom Betriebssystem und der gesetzten Sprachvariablen. Nähere Informationen dazu entnehmen Sie bitte dem jeweiligen Systemhandbuch.

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

-> reason

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

Rückgabewert

War der Aufruf erfolgreich, so liefert *t_strreason()* den Zeiger auf einen Bereich mit der Klartextdarstellung des Verbindungsabbaugrundes als String in C-Notation zurück. Wird in *reason* ein nicht definierter Wert angegeben, so liefert *t_strreason()* den Klartext: `"\t<reason> Nicht decodierbar\n"`

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

Dateien

cmxlib.cat - Meldungsdatei

Siehe auch

t_disin(), t_preason()

8.6.31 t_vdatain - Daten empfangen (data indication)

t_vdatain() nimmt ein zuvor mit *t_event()* gemeldetes Ereignis T_DATAIN entgegen. Dabei muss *t_vdatain()* vor dem nächsten *t_event()* aufgerufen werden.

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

t_vdatain() übernimmt die Daten einer empfangenen TIDU in mehrere nicht zusammenhängende Speicherbereiche. Diese Speicherbereiche werden mit Hilfe des Feldes *vdata* beschrieben.

Die Anzahl der Speicherbereiche, d. h. die Anzahl der Elemente in *vdata* wird im Parameter *vcnt* angegeben.

In *vdata* sind dann *vcnt* Strukturen *t_data* eingetragen. Jeder *t_data*-Eintrag beschreibt einen der Speicherbereiche vdata[0], vdata[1],..., vdata[vcnt-1].

Die empfangenen Daten werden nacheinander in diesen Speicherbereichen abgelegt. Dabei wird jeder Speicherbereich zuerst vollständig gefüllt, bevor zum nächsten Bereich übergegangen wird.

Zwischen zwei TIDUs derselben TSDU können beliebige andere CMX-Ereignisse für dieselbe oder eine andere Verbindung auftreten.

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

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

t_vdatain() zeigt an:

– (im Parameter *chain*)
ob noch eine weitere TIDU zur laufenden TSDU gehört

(*chain* = T_MORE) oder nicht (*chain* = T_END).

Die einzelnen TIDUs einer TSDU werden jeweils bei *t_event()* mit dem Ereignis T_DATAIN angezeigt.

– (mit dem Ergebniswert)
ob die aktuelle TIDU vollständig gelesen wurde oder nicht.

Wird der Wert T_OK zurückgegeben, so passte die TIDU in die bereitgestellten Datenbereiche hinein. Der laufende Prozess hat die aktuelle TIDU vollständig übernommen.

Wird ein Wert n > 0 zurückgegeben, so wurde nur ein Teil der TIDU gelesen. n ist die Anzahl der Byte, die aus der TIDU noch nicht gelesen wurden (Restlänge). In diesem Fall muss zunächst *t_vdatain()* oder *t_datain()* solange aufgerufen werden, bis die ganze TIDU gelesen ist. Erst dann sind wieder andere CMX-Aufrufe möglich, z. B. *t_event()*.

```
#include <cmx.h>
int   t_vdatain (const int *tref,
                 struct t_data *vdata,
                 int *vcnt,
                 int *chain);
```

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

<> vdata
: Zeiger auf einen Bereich von *t_data*-Strukturen für Datenpuffer, in die CMX die Daten der empfangenen TIDU einträgt. Folgende Struktur ist in <*cmx.h*> definiert:

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

t_datap
: Zeiger auf einen Datenbereich, in den CMX empfangene Daten der TIDU einträgt.

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

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

<- chain
: Zeiger zu einem Indikator, in dem CMX anzeigt, ob noch eine weiter TIDU zur TSDU gehört.

  Mögliche Werte:

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

T_END
: Die vorliegende TIDU ist die letzte der TSDU.

Rückgabewert

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

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

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Mindestens eine der in *vdata* angegebenen Adressen zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder für die in *tref* angezeigte Verbindung wurde kein T_DATAIN angezeigt.

T_WPARAMETER
: Der in *vcnt* angegebene Wert ist unzulässig oder mindestens eine der in *vdata* angegebenen Längen ist unzulässig.

T_COLLISION
: Für die Verbindung liegt das Ereignis T_DISIN (Verbindungsabbauanzeige) vor, wurde aber noch nicht mit *t_event()* abgefragt.

  Reaktion: *t_event()* aufrufen.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.32 t_vdatarq - Daten senden (data request)

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

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

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

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

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

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

– CMX, wenn der lokale Zwischenspeicher ausgeschöpft ist.

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

Ein erfolgreicher Abschluss von *t_vdatarq()* (T_OK) bedeutet nicht, dass die empfangende TS-Anwendung die Daten bereits entgegengenommen hat.

Ein erfolgloser Abschluss von *t_vdatarq()* (T_ERROR) bedeutet stets, dass lokal ein Fehler erkannt wurde.

```
#include <cmx.h>
int   t_vdatarq (const int *tref,
                 const struct t_data *vdata,
                 const int *vcnt,
                 const int *chain);
```

-> tref

Zeiger zu einem Feld mit der Transportreferenz der Verbindung.

-> vdata

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

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

t_datap

Zeiger auf einen Datenbereich, aus dem CMX zu sendende Daten der TIDU entnimmt.

t_datal

Für den Parameter ist die Länge des Datenbereiches *t_datap* anzugeben. Anzugeben ist mindestens 1 und höchstens die Länge einer TIDU.

-> vcnt

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

-> chain

Zeiger zu einem Indikator, der anzeigt, ob noch eine weitere TIDU zur TSDU gehört.

Mögliche Werte:

T_MORE

Es folgt noch eine weitere zur TSDU gehörende TIDU.

T_END

Die vorliegende TIDU ist die letzte der TSDU.

Rückgabewert

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

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

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Mindestens eine der in *vdata* angegebenen Adressen zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder der Prozess ist für die in *tref* angezeigte Verbindung nicht in der Datenphase oder der Datenfluss ist gesperrt.

T_WPARAMETER
: Der in *vcnt* oder *chain* angegebene Wert ist unzulässig oder mindestens eine der in *vdata* angegebenen Längen ist unzulässig oder die Summe der in *vdata* angegebenen Längen ist unzulässig.

T_COLLISION
: Für die Verbindung liegt das Ereignis T_DISIN (Verbindungsabbauanzeige) vor, wurde aber noch nicht mit *t_event()* abgefragt.

  Reaktion: *t_event()* aufrufen.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

8.6.33 t_xdatgo - Vorrangdatenanzeige freigeben (expedited data go)

t_xdatgo() gibt die gesperrte Vorrangdatenanzeige auf der angegebenen Verbindung frei. Der laufende Prozess teilt CMX damit mit, dass er wieder bereit ist, Vorrangdaten entgegenzunehmen.

Der Aufruf bewirkt im Einzelnen:

– Dem laufenden Prozess wird wieder das Ereignis T_XDATIN für die angegebene Verbindung zugestellt, sofern es vorliegt.

– Die sendende TS-Anwendung erhält (im Verlauf) das Ereignis T_XDATGO zugestellt. Sie darf wieder Vorrangdaten senden.

  Normaldaten sind von *t_xdatgo()* nicht betroffen.

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

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

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

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS kann auftreten:

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet, oder der Prozess ist für die in *tref* angegebene Verbindung nicht in der Datenphase, oder der Austausch von Vorrangdaten wurde für diese Verbindung nicht vereinbart.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

t_event(), t_error(), t_xdatstop()

8.6.34 t_xdatin - Vorrangdaten empfangen (expedited data indication)

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

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

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

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

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

<- datap
: Speicherbereich, in den CMX die Daten der empfangenen ETSDU einträgt.

<> datal
: Zeiger zu einem Feld, in das vor dem Aufruf die Länge des Datenbereiches *datap* eintragen werden muss. Anzugeben ist mindestens 1.

  Nach dem Aufruf liefert CMX in diesem Feld die Anzahl der eingetragenen Byte zurück.

Rückgabewert

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

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

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Der Zeiger *datap* zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder für die in *tref* angezeigte Verbindung wurde kein Austausch von Vorrangdaten vereinbart oder für die in *tref* angezeigte Verbindung wurde kein T_XDATIN angezeigt.

T_WPARAMETER
: Die in *datal* angegebene Länge ist unzulässig.

T_COLLISION
: Für die Verbindung liegt das Ereignis T_DISIN (Verbindungsabbauanzeige) vor, wurde aber noch nicht mit *t_event()* abgefragt.

  Reaktion: *t_event()* aufrufen.

T_CCP_END
: Das CCP nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

t_error(), t_event()

8.6.35 t_xdatrq - Vorrangdaten senden (expedited data request)

t_xdatrq() sendet auf der angegebenen Verbindung eine Expedited Transport Service Data Unit (ETSDU) mit Vorrangdaten an die empfangende TS-Anwendung. Die maximale Länge der ETSDU hängt ab vom verwendeten Transportsystem. Sie ist aber nie größer als T_EXP_SIZE Byte (gilt nicht für WAN-X25).

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

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

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

Der Vorrangdatenfluss kann gesperrt werden von:

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

– CMX,
  wenn der lokale Zwischenspeicher voll ist.

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

Ein erfolgreicher Abschluss von *t_xdatrq()* (T_OK) bedeutet nicht, dass die empfangende TS-Anwendung die Daten bereits entgegengenommen hat.

Ein erfolgloser Abschluss von *t_xdatrq()* (T_ERROR) bedeutet stets, dass lokal ein Fehler erkannt wurde.

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

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

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

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

  Minimalwert: 1

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

Rückgabewert

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

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

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_EFAULT
: Der Zeiger *datap* zeigt nicht in den Prozess-Adressraum.

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder der Prozess ist für die in *tref* angegebene Verbindung nicht in der Datenphase oder für die in *tref* angegebene Verbindung wurde kein Austausch von Vorrangdaten vereinbart oder für die in *tref* angegebene Verbindung ist der Vorrangdatenfluss gesperrt.

T_WPARAMETER
: Die in *datal* angegebene Länge ist unzulässig.

T_COLLISION
: Für die Verbindung liegt das Ereignis T_DISIN (Verbindungsabbauanzeige) vor, wurde aber noch nicht mit *t_event()* abgefragt.

  Reaktion: *t_event()* aufrufen.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

t_error(), t_event(), t_xdatstop()

8.6.36 t_xdatstop - Vorrangdatenanzeige sperren (expedited data stop)

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

t_xdatstop() bewirkt im Einzelnen:

– Der laufende Prozess teilt CMX dadurch mit, dass er bis auf weiteres nicht bereit ist, für diese Verbindung Normal- oder Vorrangdaten zu empfangen. Ein bereits angezeigtes Ereignis T_DATAIN oder T_XDATIN muss aber erst beantwortet werden.

– Der laufende Prozess bekommt die Ereignisse T_XDATIN und T_DATAIN für die angegebene Verbindung nicht mehr zugestellt. Er kann aber während die Datenanzeige gesperrt ist andere CMX-Funktionen aufrufen, z. B. eine weitere Verbindung aufbauen, abbauen oder umlenken.

– Die sendende TS-Anwendung erhält (im Verlauf) bei *t_xdatrq()* das Ergebnis T_XDATSTOP bzw. bei *t_datarq()* das Ergebnis T_DATASTOP.

  Sie darf keine Normal- oder Vorrangdaten mehr senden.

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

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

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

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

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

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

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *t_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können auftreten:

T_WSEQUENCE
: Der Prozess ist in keiner TS-Anwendung angemeldet oder der Prozess ist für die in *tref* angegebene Verbindung nicht in der Datenphase oder auf der angegebenen Verbindung ist eine TIDU oder eine ETSDU noch nicht komplett gelesen oder der Austausch von Vorrangdaten für diese Verbindung war nicht vereinbart.

T_CCP_END
: Das CCP ist nicht mehr betriebsbereit.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

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

9 Programmschnittstelle ICMX(NEA)

Dieses Kapitel beschreibt die Programmschnittstelle ICMX(NEA) zum Migrationsservice NEABX. Es enthält:

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

– Hinweise zur korrekten Verwendung der Funktionen (Zustandsautomaten),

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

Hinweise zur Verfügbarkeit der Systemoptionen für die Transportsysteme entnehmen Sie bitte der Freigabemitteilung.

9.1 Überblick über die Programmschnittstelle

Die Programmschnittstelle ICMX(NEA) realisiert den Migrationsservice NEABX. An Verarbeitungsrechnern (VAR) und Kommunikationsrechnern (KR) der TRANSDATA-Familie gibt es TS-Anwendungen, die NEA-spezifische Dienste des Transportsystems voraussetzen (z. B. UTM-Anwendungen). Solche TS-Anwendungen setzen also Dienste voraus, die über die Funktionalität eines ISO-Transportsystems der Norm ISO 8072 hinausgehen. Mit Hilfe des Migrationsservices NEABX ist es einer TS-Anwendung an Ihrem System (CMX-Anwendungen) möglich, mit solchen TS-Anwendungen an einem VAR oder KR zu kommunizieren, ohne dass die TS-Anwendung am fernen System ihre Kommunikationsschnittstelle ändert. D. h. ICMX(NEA) bietet Funktionen, die die Differenz zwischen den erwarteten Leistungen und den von einem ISO-Transportsystem gebotenen Leistungen ausgleichen.

Insbesondere werden mit Hilfe der ICMX(NEA)-Funktionen in der Verbindungsaufbauphase das NEABV-Protokoll und in der Datenphase das NEABX-Protokoll aufgebaut, ausgetauscht und ausgewertet.

ICMX(NEA) ist realisiert als eine Menge von C-Funktionen. Intern setzt NEABX jede ICMX(NEA)-Funktion in eine oder mehrere ICMX(L)-Funktionen um.

ICMX(NEA) - Schnittstelle zum verbindungsorientierten Transport Service

NEABX bietet mit ICMX(NEA) eine Programmschnittstelle zum verbindungsorientierten Transport Service (TS). Dieser TS erlaubt es zwei Anwendungen (TS-Anwendungen), Nachrichten über eine Transportverbindung (TV) auszutauschen. Bei der Kommunikation über einen verbindungsorientierten TS werden die Nachrichten verlust- und duplikatfrei ausgetauscht. Die Nachrichtenreihenfolge wird beibehalten. Ist eine TV aufgebaut, so wird ihr in beiden Endsystemen je eine Transportreferenz (tref) zugeordnet. Die Transportreferenzen identifizieren die TV eindeutig. In der Datenphase kann man somit auf die Übertragung und Verarbeitung von Adressen verzichten. Parameter, die den Transport der Nachrichten auf der TV beeinflussen, können von den TS-Anwendungen beim Verbindungsaufbau verhandelt werden. Zum korrekten Ablauf der Kommunikation müssen Regeln beachtet werden, die im Folgenden beschrieben sind.

ICMX(NEA) macht 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.

Intern ist jeder TV eine exklusiv eröffnete 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.

Namen und Adressen

Jede TS-Anwendung hat einen GLOBALEN NAMEN. Unter diesem Namen ist die TS-Anwendung im Netz eindeutig identifizierbar. Die Vergabe der GLOBALEN NAMEN erfolgt durch die Administration. Sie muss sicherstellen, dass die Namen aller TS-Anwendungen voneinander verschieden sind.

Die TS-Anwendung arbeitet ausschließlich mit GLOBALEN NAMEN. Bei der Anmeldung gibt sie ihren GLOBALEN NAMEN an und beim Verbindungsaufbau den GLOBALEN NAMEN des Kommunikationspartners.

Der TNSX, ein Bestandteil von CMX, übersetzt die GLOBALEN NAMEN in den LOKALEN NAMEN der lokalen TS-Anwendung und die TRANSPORTADRESSE der fernen TS-Anwendung und umgekehrt.

Zur Abfrage des LOKALEN NAMENS der TS-Anwendung und der TRANSPORTADRESSE des Kommunikationspartners stehen an der Schnittstelle ICMX(L) die Funktionen *t_getloc()* und *t_getaddr()* zur Verfügung.

In <*neabx.h*> sind die Strukturen *x_myname* und *x_partaddr* definiert.

In *x_myname* übernimmt (übergibt) die TS-Anwendung ihren LOKALEN NAMEN vom (an den) TNSX, in *x_partaddr* die TRANSPORTADRESSE.

Die Strukturen sind wie folgt aufgebaut:

```
struct x_myname {
    char x_mnmode;          /* = X_MNMODE */
    char x_mnres;           /* = 0 */
    short x_mnln            /* Länge des ausgefüllten Teils von
                             x_myname*/
    char x_mn[X_MNSIZE];    /* Feld für die T-Selektoren des */
                            /* LOKALEN NAMENS */
 }

struct x_partaddr {
    char x_pamode;          /* = X_PAMODE */
    char x_pares;           /* = 0 */
    short x_palng;          /* Länge des ausgefüllten Teils von
                             x_partaddr */
    char x_pa[X_PASIZE];    /* Feld für die Partneradresse */
 }
```

Die Elemente der Struktur *x_myname* haben die folgende Bedeutung:

x_mnmode = X_MNMODE
: gibt an, dass das Feld *x_mn* einen LOKALEN NAMEN enthält.

x_mnres, *x_mn*[X_MNSIZE]
: sind für Sie nicht relevant. Der Inhalt dieser Felder wird nur vom TNSX übernommen und an NEABX weitergereicht.

x_mnlng
: gibt die Länge aller in der Struktur *x_myname* übergebenen Daten an.

Die Elemente der Struktur *x_partaddr* haben die folgende Bedeutung:

x_pamode = X_PAMODE
: gibt an, dass das Feld *x_pa* eine TRANSPORTADRESSE enthält.

x_pares, *x_pa*[X_PASIZE]
: sind für Sie nicht relevant. Der Inhalt dieser Felder wird nur vom TNSX übernommen und an NEABX weitergereicht.

x_palng
: gibt die Länge aller in der Struktur *x_partaddr* übergebenen Daten an. LOKALER NAME und TRANSPORTADRESSE werden in der Union *x_address* an NEABX übergeben oder von NEABX übernommen.

```
union x_address {
struct x_myname xmyname;
struct x_partaddr xpartaddr;
}
```

Fehlerbehandlung und -diagnose

Alle Funktionsaufrufe enden mit einer Rückmeldung. Diese zeigt zum Beispiel mit T_OK den erfolgreichen Abschluss an. Tritt ein Fehler auf, so liefert die Funktion pauschal den Wert X_ERROR zurück. Anschließend können Sie mit der Funktion *x_error()* detailliertere Diagnoseinformationen abfragen. *x_error()* müssen Sie sofort nach dem Auftreten des Fehlers aufrufen.

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.

Wie diese Fehlercodes aufgebaut sind, ist in Kapitel „Ereignisverarbeitung und Fehlerbehandlung“ auf Seite 37 beschrieben.

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. Tritt hier ein Fehler auf, so wird die Verbindung abgebaut und ein Grund für den Abbau an CMX übergeben. Diesen Verbindungsabbaugrund erhält die TS-Anwendung, wenn sie die Funktion *x_disin()* aufruft.

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

x_strerror()
: liefert zu einem von ICMX(NEA) erhaltenen Fehlercode den Zeiger auf den Klartextstring.

x_perror()
: ermittelt zu einem von ICMX(NEA) erhaltenen Fehlercode den Klartextstring mit *x_strerror()* und schreibt ihn nach *stderr*.

  Mit den ICMX(L)-Funktionen *t_strreason()* und *t_preason()* können Sie die beim *x_disin()* erhaltenen Codes für einen Verbindungsabbaugrund in Klartext übersetzen.

Fehlercode und Verbindungsabbaugrund können auf Kommandoebene mit *cmxdec* in Klartext aufbereitet werden.

Zur Diagnose steht Ihnen an ICMX(NEA) ein Verfolger zur Verfügung. Er ist über die Environmentvariable NEATRACE einstellbar. Der Verfolger protokolliert die Aufrufe mit ihren Argumenten kompakt in temporäre Dateien. Das Aufbereitungsprogramm *neal* setzt dann entkoppelt das Protokoll in Klartext um (siehe Handbücher „CMX, Betrieb und Administration“ [1] und [2]).

TS-Anwendungen, Transportverbindungen und 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 Prozess-Konzept des Systems 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 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, d. h. verschiedene 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 NEABX sind prinzipiell asynchron ausgelegt, das heißt, nach Absetzen eines Aufrufes muss die TS-Anwendung nicht auf die eventuelle Antwort aus dem Netz (TS-Ereignis) 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-Ereignis sofort verarbeiten kann. Der Wartezustand kann durch Angabe einer Wartezeit zeitlich begrenzt werden oder durch Signale wie SIGALARM vorzeitig abgebrochen

werden. In beiden Fällen setzt NEABX den Prozess mit dem TS-Ergebnis X_NOEVENT fort. Der synchrone Mechanismus ist nützlich für TS-Anwendungen, die zwischen zwei TS-Ereignissen 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, dem Ablauf der Wartezeit oder dem Eintreffen eines Signals suspendiert. Der suspendierte Prozess wird beim Eintreffen eines Signals geweckt. *x_event()* kehrt mit X_NOEVENT oder X_ERROR zurück. Bei Ablauf der Wartezeit setzt sich der Prozess mit dem TS-Ereignis X_NOEVENT fort. 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.

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 oder Ablauf der Wartezeit.

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 eintreffenden Verbindungsumlenkung. (Dieses TS-Ereignis ist lokal, es ist eine Erweiterung des TS's zur Flexibilisierung der Implementierung von TS-Anwendungen.)

X_DATAIN
: Von einer TS-Anwendung gesendete Normaldaten sind eingetroffen.

X_XDATIN
: Von einer TS-Anwendung gesendete Vorrangdaten sind eingetroffen.

X_DATAGO
: Eine durch die Fluss-Regelung gesetzte Sendesperre für Normaldaten wird aufgehoben.

X_XDATGO
: Eine durch die Fluss-Regelung 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.

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

TS-Ereignis	abholende Funktion
X_CONCF	x_concf()
X_CONIN	x_conin()
X_DATAIN	x_datain()
X_DISIN	x_disin()
X_ERROR	x_error()
X_REDIN	x_redin()
X_REPCRQ	x_conrq()
X_REPCCF	x_concf()
X_REPCIN	x_conin()
X_XDATIN	x_xdatin()

Tabelle 9: TS-Ereignisse und zugehörige Funktionen

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.

Anmelden/Abmelden bei NEABX

Die Kommunikation einer TS-Anwendung über NEABX wird aktiviert, indem sich der erste Prozess für sie bei NEABX anmeldet. Bei der Anmeldung wird exklusiv für diesen Prozess eine Gerätedatei eröffnet. Über diese Gerätedatei werden Aufträge zwischen den NEABX-Bibliotheksfunktionen und dem Betriebssystem ausgetauscht. Durch den ersten Prozess, der sich für eine TS-Anwendung anmeldet, wird diese TS-Anwendung erzeugt. Dabei wird ein Dienstzugriffspunkt (Transport Service Access Point TSAP) eingerichtet, an dem der TS zur Verfügung steht. Mit der Anmeldung des ersten Prozesses wird die TS-Anwendung an diesen TSAP gebunden. Dem TSAP wird der LOKALE NAME der TS-Anwendung zugeordnet, unter dem die TS-Anwendung in diesem Endsystem erreichbar ist. Sie wird damit aus dem Netz adressierbar. Bei der Abmeldung werden noch bestehende TVen und der TSAP abgebaut, die Prozess-Umgebung aufgelöst und belegte Betriebsmittel für zukünftige Verwendung freigegeben.

Derselbe Prozess kann sich gleichzeitig für mehrere TS-Anwendungen anmelden (d. h. mehrere TSAP verwalten) und in jeder dieser TS-Anwendungen mehrere Verbindungsendpunkte (Transport Connection Endpoint TCEP) unterhalten. Es können sich auch mehrere Prozesse in derselben TS-Anwendung anmelden (denselben TSAP verwenden) und in jeder aktiv TVen aufbauen oder passiv auf Verbindungsanzeigen warten ohne miteinander zu interferieren. Allerdings ist jeder TCEP genau 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 (den laufenden Prozess) einer 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 (den laufenden Prozess) einer TS-Anwendung bei NEABX 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 NEABX nicht mehr bekannt.

Verbindungsaufbau, -abbau und -umlenkung

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 den Datentransfer und können Benutzerdaten austauschen.

Jede der TS-Anwendungen sowie NEABX können jederzeit die TV abbauen. 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 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 Abschnitt „Zustandsautomaten“ auf Seite 233) 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. Die TV ist dann für den abgebenden Prozess nicht mehr verfügbar.

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).

  Beim Aufruf von *x_conrq()* muss das NEABV-Protokoll in Form von Benutzerdaten übertragen werden. Die Benutzerdaten sind in einer Optionsstruktur an NEABX zu übergeben.

  Liefert *x_conrq()* den Wert X_REPEAT zurück, so muss *x_conrq()* mit gleichen Parametern nach Eintreffen des Ereignisses X_REPCRQ noch einmal aufgerufen werden. Die Benutzerdaten konnten noch nicht vollständig übertragen werden.

x_conin()
: übernimmt von NEABX die mit X_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.

  Beim Verbindungsaufbau wird das NEABV-Protokoll in Form von Benutzerdaten übertragen. Die gerufene TS-Anwendung muss die Benutzerdaten in einer Optionsstruktur von NEABX entgegennehmen.

Liefert *x_conin()* den Wert X_REPEAT zurück, so muss *x_conin()* nach Eintreffen des Ereignisses X_REPCIN mit gleichen Parametern wiederholt werden. Die Benutzerdaten sind noch nicht vollständig empfangen worden.

x_conrs()
: beantwortet (akzeptiert) die Verbindungsanforderung, nachdem sie mit X_CONIN angezeigt und von NEABX übernommen wurde.

  Beim Aufruf von *x_conrs()* muss das NEABV-Protokoll in Form von Benutzerdaten übertragen werden. Die Benutzerdaten sind in einer Optionsstruktur an NEABX zu übergeben.

x_concf()
: übernimmt von NEABX die mit X_CONCF angezeigte Antwort der gerufenen TS-Anwendung. Damit ist der Verbindungsaufbau vollzogen. Beim Verbindungsaufbau wird das NEABV-Protokoll in Form von Benutzerdaten übertragen. Die TS-Anwendung muss die Benutzerdaten in einer Optionsstruktur von NEABX entgegennehmen.

  Liefert *x_concf()* den Wert X_REPEAT zurück, so muss *x_concf()* nach Eintreffen des Ereignisses X_REPCCF mit gleichen Parametern wiederholt werden. Die Benutzerdaten sind noch nicht vollständig empfangen worden.

x_disrq()
: fordert den Abbau einer Verbindung an. Die Funktion kann jederzeit von jeder TS-Anwendung 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. Beim Aufruf der Funktion wird der TS-Anwendung auch der Verbindungsabbaugrund übergeben.

x_redrq()
: lenkt die TV an einen Prozess derselben TS-Anwendung um. Dem abgegebenen Prozess ist sie anschließend nicht mehr verfügbar.

  Beim Aufruf von *x_redrq()* muss eine Optionsstruktur angegeben werden. Der Migrationsservice des abgebenden Prozesses übergibt darin Informationen an den Migrationsservice des empfangenden Prozesses.

  Liefert *x_redrq()* den Wert X_IMPOSSIBLE zurück, so kann die Verbindungsumlenkung nicht ausgeführt werden.

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.

Beim Aufruf von *x_redin()* muss eine Optionsstruktur angegeben werden. Der Migrationsservice des empfangenden Prozesses empfängt darin Informationen vom Migrationsservice des abgebenden Prozesses.

Datenaustausch, Fluss-Regelung

Nachdem eine TV aufgebaut ist, kann über sie der Transfer von Normaldaten und (optional) Vorrangdaten erfolgen. Der Transfer findet nachrichtenorientiert statt: die TS-Anwendungen tauschen Transport Service Data Units (TSDU), das sind Nachrichten beliebiger Länge, oder Expedited Transport Service Data Units (ETSDU), das sind Vorrangdaten beschränkter Länge, aus. Vorrangdaten sind auf wenige Byte beschränkt, sie werden mit Vorrang zum Strom der Normaldaten übertragen und in eigenen Warteschlangen geführt. NEABX garantiert nur, dass Vorrangdaten nie später als danach versendete Normaldaten bei der empfangenden TS-Anwendung eintreffen. Pro Aufruf kann an NEABX höchstens eine komplette ETSDU übertragen werden.

Die Übergabe einer Nachricht an NEABX erfolgt in Portionen der Länge einer Transport Interface Data Unit (TIDU). Die Länge der TIDU ist TV-spezifisch, sie muss deshalb für jede TV von NEABX abgefragt werden (*x_info()*). Ist die Nachricht länger als eine TIDU, so muss die Nachricht mit mehreren Sendeaufrufen übertragen werden. Ein Parameter beim Sendeaufruf zeigt an, ob eine weitere TIDU für diese Nachricht folgt (X_MORE) oder nicht (X_END). Daraus ist nicht ableitbar, wie die TIDU für die Übertragung oder Zustellung zur empfangenden TS-Anwendung verpackt wird. NEABX garantiert nur, dass die sequentielle Zusammenfügung der TIDUs auf Empfangsseite wieder die Nachricht der Sendeseite liefert. Die TIDU-Länge kann für beide TS-Anwendungen verschieden sein und hängt von der TV ab. NEABX garantiert nicht, dass bei der empfangenden TS-Anwendung jede außer der letzten TIDU einer Nachricht maximal gefüllt zugestellt wird.

Die empfangende TS-Anwendung erhält die Ankunft einer TIDU oder einer ETSDU über das TS-Ereignis X_DATAIN bzw. X_XDATIN angezeigt. Sie holt dann die TIDU bzw. ETSDU mit einem entsprechenden Funktionsaufruf vollständig ab. Als Zusatzinformation liefert die Funktion *x_event()* in der Optionsstruktur die Länge der Daten.

Die Übertragung von TIDUs und ETSDUs unterliegt Fluss-Regelungsmechanismen, die von NEABX und den TS-Anwendungen geregelt werden können. Die Rückmeldung X_DATASTOP oder X_XDATSTOP beim Senden sagt der sendenden TS-Anwendung, dass die TIDU bzw. die ETSDU erfolgreich verarbeitet, der Fluss der TIDUs (ETSDUs) aber gesperrt wurde. Es kann keine TIDU (ETSDU) mehr gesendet werden, bis der Fluss wieder freigegeben wird. Dies wird durch das TS-Ereignis X_DATAGO (X_XDATGO) angezeigt.

Die empfangende TS-Anwendung sperrt und entsperrt den Fluss der TIDUs und ETSDUs durch Funktionsaufrufe an NEABX, die sich für die sendende TS-Anwendung wie oben beschrieben, auswirken.

Die folgenden Funktionen realisieren Datenaustausch und (aktive) Fluss-Regelung:

x_datarq()
: fordert die Übertragung einer (evtl. teilgefüllten) TIDU an. Die Rückmeldung X_DATASTOP besagt, dass der Datenfluss gesperrt ist. Weitere Sendeanforderungen werden solange mit Fehler abgewiesen, bis der Datenfluss wieder freigegeben wird.

  Falls beim Verbindungsaufbau der Austausch des NEABX-Protokolls in der Datenphase vereinbart wurde, muss bei der **ersten** TIDU einer TSDU eine Optionsstruktur zum Austausch des NEABX-Protokolls angegeben werden. Bei allen weiteren TIDUs der TSDU ist die Angabe einer Optionsstruktur **nicht zulässig**. Wurde der Austausch des NEABX-Protokolls nicht vereinbart, so darf **keine** Optionsstruktur angegeben werden.

x_datain()
: übernimmt die Daten einer TIDU von NEABX, nachdem die TIDU mit X_DATAIN angezeigt wurde. Falls beim Verbindungsaufbau der Austausch des NEABX-Protokolls in der Datenphase vereinbart wurde, muss beim Empfangen der **ersten** TIDU einer TSDU eine Optionsstruktur zum Austausch des NEABX-Protokolls angegeben werden. Bei allen weiteren TIDUs der TSDU ist die Angabe einer Optionsstruktur **nicht zulässig**. Wurde der Austausch des NEABX-Protokolls nicht vereinbart, so darf **keine** Optionsstruktur angegeben werden.

x_xdatrq()
: fordert die Übertragung einer ETSDU mit Vorrangdaten an. Die Rückmeldung X_XDATSTOP besagt, dass der Fluss der ETSDUs gesperrt ist. Weitere Sendeanforderungen werden solange mit Fehler abgewiesen, bis der Fluss wieder freigegeben wird. Falls beim Verbindungsaufbau der Austausch des NEABX-Protokolls in der Datenphase vereinbart wurde, muss eine Optionsstruktur zum Austausch des NEABX-Protokolls angegeben werden. Wurde der Austausch des NEABX-Protokolls nicht vereinbart, so darf **keine** Optionsstruktur angegeben werden.

x_xdatin()
: übernimmt die Vorrangdaten von NEABX, nachdem sie mit X_XDATIN angezeigt wurden. Falls beim Verbindungsaufbau der Austausch des NEABX-Protokolls in der Datenphase vereinbart wurde, muss beim Empfangen der ETSDU eine Optionsstruktur zum Austausch des NEABX-Protokolls angegeben werden. Wurde der Austausch des NEABX-Protokolls nicht vereinbart, so darf **keine** Optionsstruktur angegeben werden.

Informationsdienst

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

x_info()
: liefert für eine eingerichtete TV die Länge der TIDU. Die TIDU steht in der Regel erst dann fest, wenn der Verbindungsaufbau vollständig abgeschlossen worden ist.

9.2 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 gerahmtes Rechteck dargestellt ist (siehe Bild „Auf- und Abbau und Umlenkung von TVen“ auf Seite 235), 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.

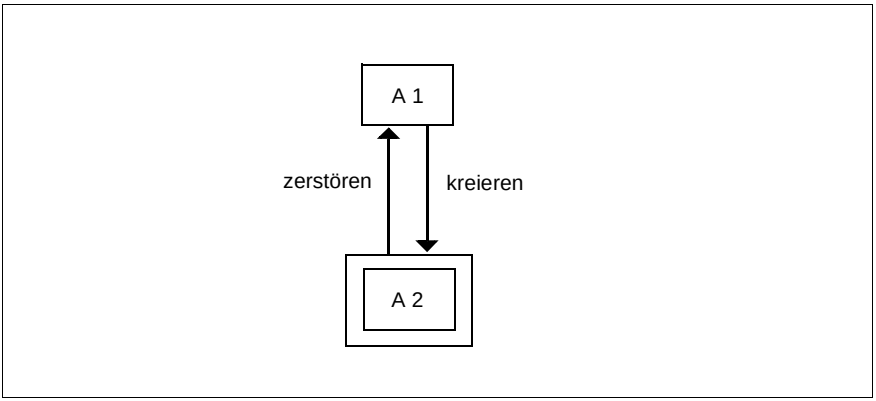

Bild 30: Aktivierung/Deaktivierung

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.

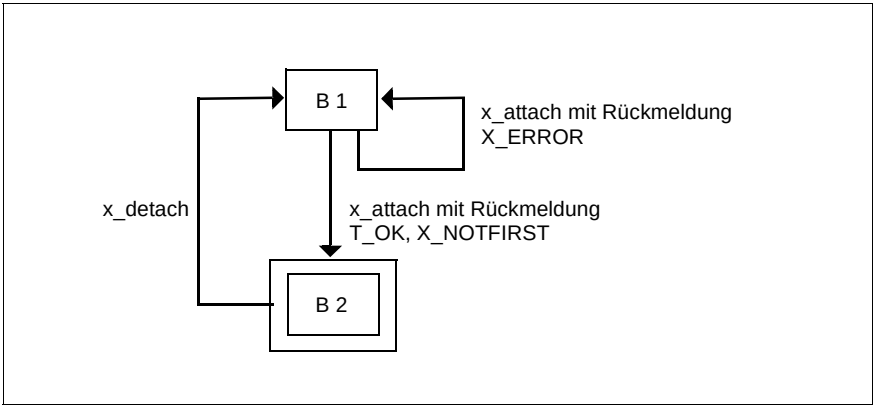

Bild 31: An-/Abmeldung von TS-Anwendungen

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.

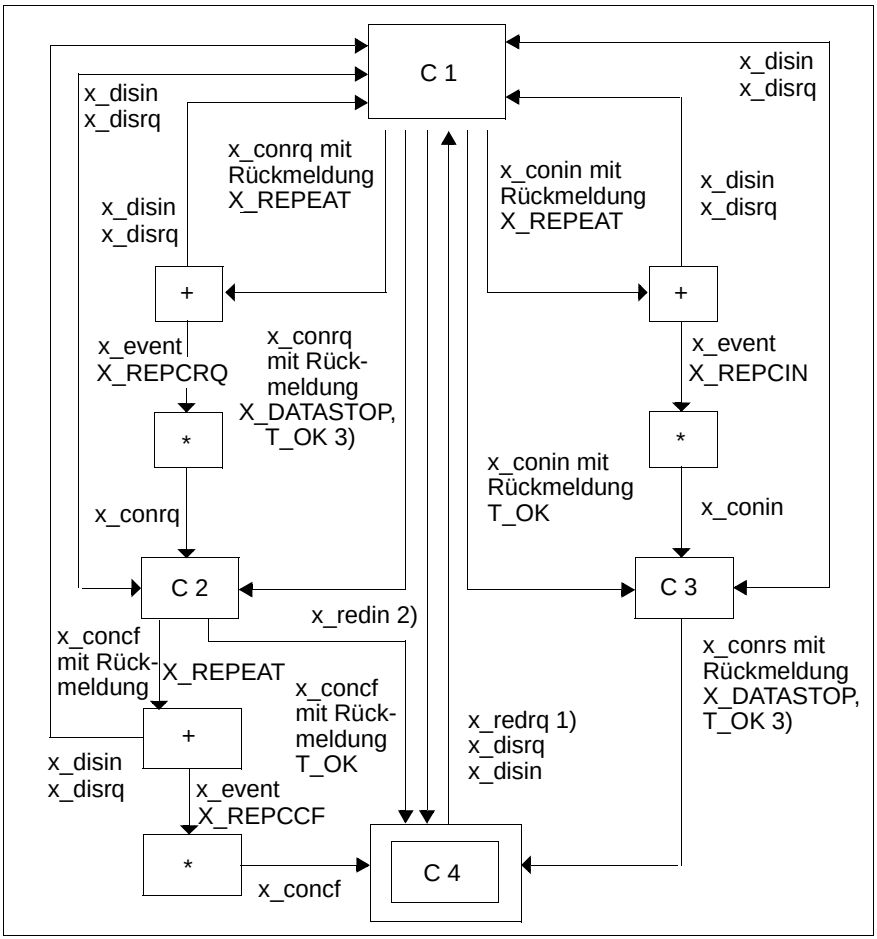

Bild 32: Auf- und Abbau und Umlenkung von TVen

1. *x_redrq()* ist nur dann erlaubt, wenn der entsprechende Datensendeautomat in D1 und die Datenempfangsautomaten in D1 und D4 sind.
2. *_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 T_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

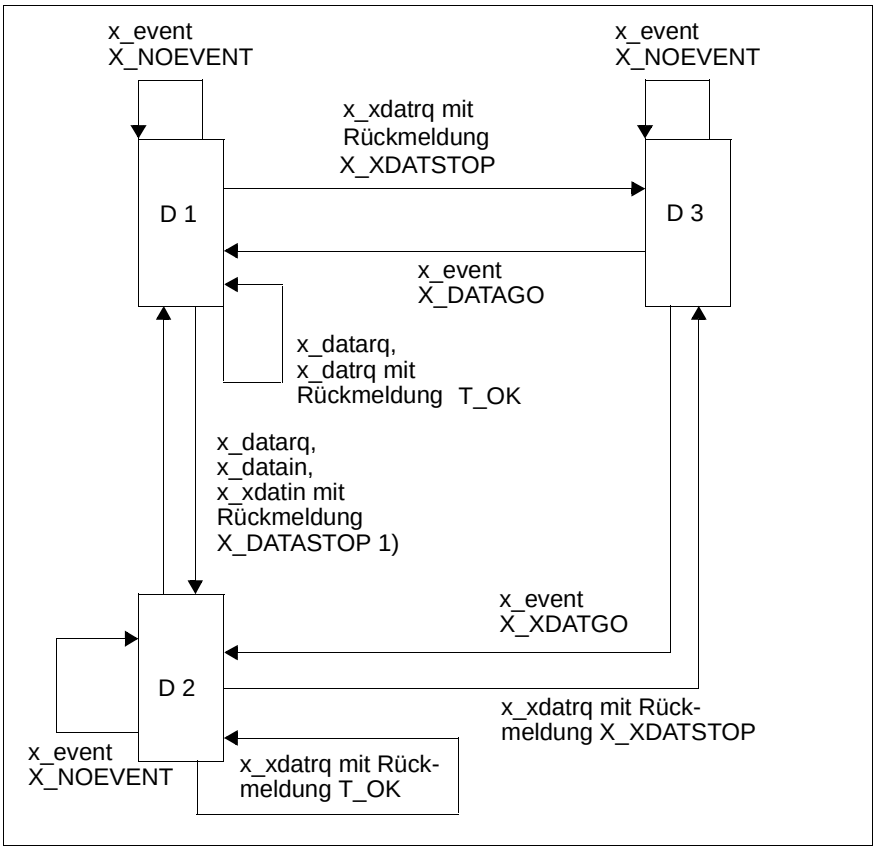

Bild 33: Datensendeautomat für Normal- und Vorrangdaten

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 Fluss-Regelung möglich.

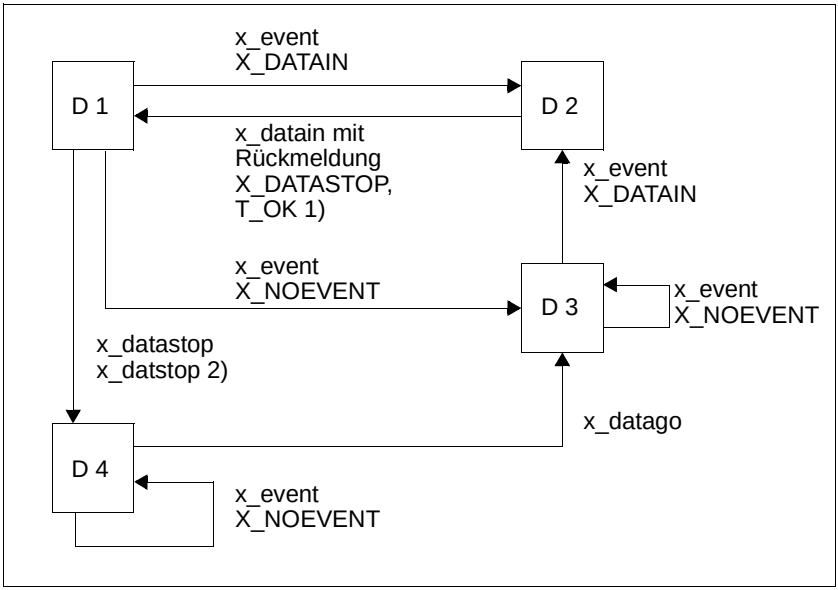

Bild 34: Datenempfangsautomat für Normaldaten

2. Bei Rückmeldung X_DATASTOP geht der zugehörige Datensendeautomat in den Zustand D2 über.
3. *_xdatstop()* ist nur erlaubt, wenn der Datenempfangsautomat für Vorrangdaten in D1 ist.

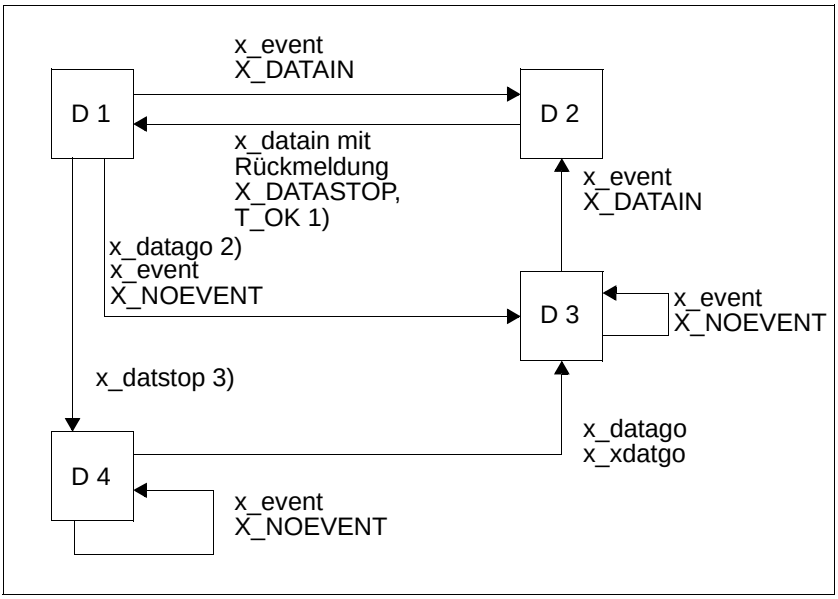

Bild 35: Datenempfangsautomat für Vorrangdaten

4. Bei Rückmeldung X_DATASTOP geht der zugehörige Datensendeautomat in den Zustand D2 über.
5. *x_datago()* ist nur erlaubt, wenn der Datenempfangsautomat für Normaldaten in D4 ist.
6. *x_xdatstop()* ist nur erlaubt, wenn der Datenempfangsautomat für Normaldaten in D1 ist.

9.3 NEABV-Protokoll

9.3.1 Das NEABV-Protokoll für die Kommunikation über ICMX(NEA)

Soll Ihre TS-Anwendung mit einer TS-Anwendung kommunizieren, die TRANSDATA-spezifische Funktionen des Transportprotokolls voraussetzen, so müssen Sie beim Verbindungsaufbau über die Migrationsschnittstelle ICMX(NEA) das Benutzerdienst-Verbindungsprotokoll (NEABV, Fujitsu Siemens Computers Norm SN 77303) einhalten.

Kommunikationspartner, bei denen Sie das NEABV-Protokoll einhalten müssen, 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 TS-Anwendung das NEABV-Protokoll vor die eigentliche Benutzernachricht (Benutzerverbindungsnachricht) in den Datenpuffer (*x_udatap*) eintragen.

Das NEABV-Protokoll können Sie auch mit Hilfe der NEABX-Dienstfunktion *x_neavo()* erzeugen.

Bei den Aufrufen *x_conin()* und *x_concf()* steht das NEABV-Protokoll als Benutzerdaten im Datenpuffer (*x_udatap*).

Das NEABV-Protokoll können Sie mit Hilfe der NEABX-Dienstfunktion *x_neavi()* analysieren.

Im Folgenden ist der Aufbau des NEABV-Protokolls bei Rechnerkopplung über LAN und WAN beschrieben.

Aufbau der Benutzerdaten im Datenpuffer x_udatap bei Rechnerkopplung

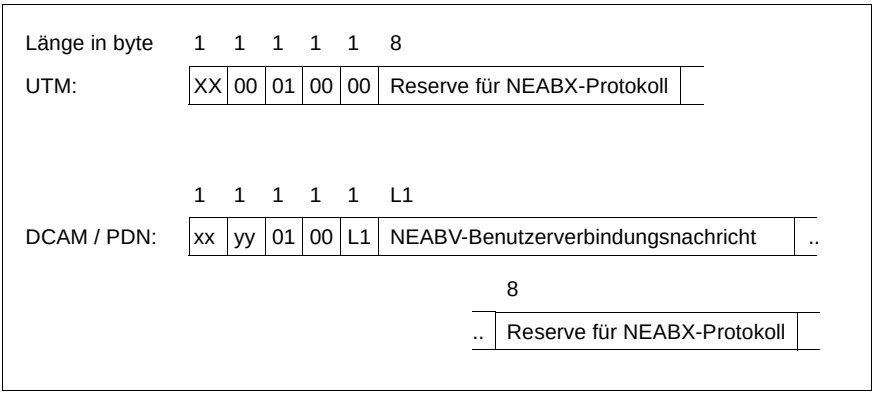

Bild 36: Aufbau des Datenpuffers bei Rechnerkopplung

Die einzelnen Elemente des NEABV-Protokolls haben folgende Bedeutung und Werte:

xx

Vereinbarung über den Austausch des NEABX-Protokolls in der Datenphase.

x_conrq(), *x_conrs()*, *x_concf()*: xx = X'01'

D. h. kein NEABX-Protokoll in der Datenphase.

x_conin(): xx = X'00' oder X'01'

D. h. die Partneranwendung richtet sich nach der Festlegung durch die NEABX-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 Benutzerverbindungsnachricht.

Es ist i.a.: X'00' <= L1 <= X'50' d. h. die folgende Benutzerverbindungsnachricht kann zwischen 0 und 80 Zeichen lang sein.

Bei der Rechnerkopplung über WAN darf die Benutzer-Verbindungsnachricht jedoch maximal 79 Zeichen lang sein.

9.3.2 Die NEABX-Dienstfunktionen (NEABV-Service)

Dem Benutzer wird ein Dienst zur Verfügung gestellt, durch den das NEABV-Protokoll erzeugt wird, um Fehlbedienung bei der Erstellung des Protokolls zu vermeiden.

Das NEABV-Protokoll ist einzufügen bei:

– der Verbindungsanforderung mit *x_conrq()*

– der Bestätigung der Verbindungsanforderung mit *x_conrs()*

und wird geliefert bei:

– der Entgegennahme einer Verbindungsanforderung mit *x_conin()*

– der Herstellung der Verbindung mit *x_concf()*

Die zwei nachfolgenden Aufrufe dienen zur Erzeugung und zur Analyse des NEABV-Protokolls.

x_neavo()

erzeugt das NEABV-Protokoll bei Ausgabe. *x_neavo* kann vor *x_conrq()* und *x_conrs()* aufgerufen werden.

x_neavi()

analysiert ein ankommendes NEABV-Protokoll. *x_neavi()* kann nach *x_conin()* und *x_concf()* aufgerufen werden, um das von der Partner-TS-Anwendung eingetroffene NEABV-Protokoll zu analysieren.

Verwendung des Parameters x_init bei den Aufrufen x_neavi() und x_neavo()

Über den Parameter *x_init* handeln die Kommunikationspartner aus, wer in der Datenphase mit der Datenübermittlung beginnen soll. Im Folgenden ist beschrieben, was bei der Versorgung des Parameters zu beachten ist.

Mögliche Werte von *x_init* sind:

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.*)

(Die mit *) gekennzeichneten Werte sind gleichcodiert.)

- Angaben für *x_init* bei Rechnerkopplung:

  Hier sind die möglichen *x_init*-Angaben der rufenden TS-Anwendung und die erwarteten Antworten der gerufenen TS-Anwendung beschrieben. Die rufende TS-Anwendung übergibt den Vorschlag beim *t_conrq()* an die gerufene TS-Anwendung, die ihn beim Aufruf *x_conin()* empfängt. Die Antwort *x_init* wird von der gerufenen TS-Anwendung in *x_conrs()* gegeben und in *x_concf* von der rufenden TS-Anwendung empfangen.

  – *x_init* = X_MYINIT

    Vorschlag der rufenden TS-Anwendung, dass sie mit der Datenübermittlung beginnen will.

    Antwort der gerufenen TS-Anwendung:

    x_init = X_INITOK (Einverständnis mit Partnervorschlag)

    oder *x_disrq()* im Falle keines Einverständnisses.

  – *x_init* = X_INITRQ

    Die rufende TS-Anwendung wartet auf den Vorschlag der gerufenen TS-Anwendung.

    Erwartete Antwort der gerufenen TS-Anwendung:

    x_init = X_MYINIT (die gerufene TS-Anwendung beginnt mit der Datenübermittlung)

    oder *x_disrq()*.

    Wird eine andere Antwort an die rufende TS-Anwendung übergeben, so sollte diese 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. langer Wartezeit. Deshalb sollte dann mit *x_disrq()* geantwortet werden.

9.4 Transportsystem-spezifische Besonderheiten

Die in Abschnitt „Transportsystem-spezifische Besonderheiten“ auf Seite 104 beschriebenen transportsystem-spezifischen Besonderheiten treffen auch für TS-Anwendungen an ICMX(NEA) zu. Die beschriebenen Besonderheiten beziehen sich auf die entsprechenden Funktionsaufrufe mit Präfix x_ und CMX-Events mit Präfix X_.

9.5 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 TNSX 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:
```

```
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:
            :
}
```

```
            continue;

        case X_ERROR:
            x_detach();
            exit();
        default:
            :
}
```

9.6 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 Präprozessordefinitionen, die mit „T_“, „X_“ oder „TS“ beginnen, sind für NEABX reserviert.
4. Auf Anforderung des Benutzers werden von den CMX-Komponenten im Betriebssystemkern Signale verschickt und in der NEABX-Bibliothek eingefangen (in der Regel SIGIO und/oder SIGTERM). Die Verwendung von benutzereigenen Signalroutinen sollte daher sorgfältig programmiert werden.

9.7 ICMX(NEA) - Funktionsaufrufe

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

Folgende Kennzeichnungen werden bei der Parameterbeschreibung verwendet:

->
: kennzeichnet Parameter, in denen NEABX einen vom Aufruf bereitgestellten Wert erwartet.

<-
: kennzeichnet Parameter, in denen NEABX nach dem Aufruf einen Wert liefert.

<>
: kennzeichnet Parameter, in denen der Aufrufer einen Wert bereitstellen muss, der dann von NEABX modifiziert wird.

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

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

9.7.1 x_attach - Anmelden eines Prozesses bei NEABX (attach process)

x_attach() meldet den laufenden Prozess bei NEABX an. Durch die Parameter, die beim Aufruf von *x_attach()* übergeben werden, legt man fest:

– für welche TS-Anwendung sich der Prozess anmeldet,

– welche Arten des Verbindungsaufbaus (passiv, aktiv usw.) dem Prozess in dieser TS-Anwendung möglich sind,

– wie viele Verbindungen der Prozess in dieser TS-Anwendung gleichzeitig haben darf.

Die TS-Anwendung, für die sich der Prozess anmeldet, hat einen netzweit eindeutigen GLOBALEN NAMEN und einen oder mehrere T-Selektoren, die im lokalen System eindeutig sind. Die T-Selektoren werden zusammengefasst zum LOKALEN NAMEN. Der LOKALE NAME muss NEABX als Parameter übergeben werden. Mit Hilfe des ICMX(L)-Aufrufs *t_getloc()* und des GLOBALEN NAMENS der TS-Anwendung kann der LOKALE NAME vom TNSX abgefragt und in einem Datenbereich bereitgestellt werden.

Der laufende Prozess kann sich durch wiederholte *x_attach()*-Aufrufe für mehrere verschiedene TS-Anwendungen bei NEABX anmelden.

Ebenso können sich mehrere Prozesse für dieselbe TS-Anwendung, d. h. mit demselben LOKALEN NAMEN bei NEABX anmelden. Der erste Prozess, der sich für eine TS-Anwendung anmeldet, erzeugt diese TS-Anwendung. Will man den gleichen Prozess für die Programmschnittstellen ICMX(L) und ICMX(NEA) anmelden, so muss man *t_attach()* und *x_attach()* mit verschiedenen LOKALEN NAMEN aufrufen.

NEABX nimmt Verbindungswünsche für eine TS-Anwendung aus dem Netz entgegen, sobald sich ein Prozess in dieser TS-Anwendung für die Annahme von Verbindungsanzeigen bei NEABX angemeldet hat, d. h. wenn man X_PASSIVE in *x_apmode* gesetzt hat.

Haben sich mehrere Prozesse für eine TS-Anwendung mit X_PASSIVE angemeldet, so stellt NEABX alle Verbindungsanzeigen für diese TS-Anwendung zunächst dem Prozess zu, der sich als erster mit X_PASSIVE für diese TS-Anwendung angemeldet hat. Erst wenn die Anzahl der Verbindungen erreicht ist, die dieser Prozess für diese TS-Anwendung haben darf, werden die eintreffenden Verbindungsanzeigen einem der anderen Prozesse zugestellt. Die Reihenfolge hierfür ist nicht definiert.

```
#include <cmx.h>
#include <neabx.h>
int   x_attach (struct x_myname *name,
                struct x_opta1 *x_opt);
```

-> name

Zeiger auf eine Struktur *x_myname*, in der der LOKALE NAME der TS-Anwendung zu übergeben ist. Den LOKALEN NAMEN liefert der TNSX als Eigenschaft zum GLOBALEN NAMEN.

-> x_opt

x_opt ist der Zeiger auf die Struktur *x_opta1* oder NULL. Wenn Sie NULL angeben, setzt NEABX die angegebenen Standardwerte.

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

x_apmode legt fest, welche Arten des Verbingungsaufbaus diesem Prozess in dieser TS-Anwendung möglich sind.

Folgende Werte können Sie angeben:

X_ACTIVE

Der Prozess soll aktiv Verbindungen aufbauen.

X_PASSIVE

Der Prozess soll passiv auf Verbindungswünsche warten.

X_REDIRECT

Der Prozess soll umgelenkte Verbindungen annehmen.

Diese Werte können Sie mit bitweisem ODER (|) kombinieren, z. B.

X_ACTIVE | X_PASSIVE.

Standardwert bei Angabe von NULL:

X_ACTIVE | X_PASSIVE | X_REDIRECT.

x_conlim

Maximalzahl der Verbindungen, die dieser Prozess pro Anwendung gleichzeitig haben darf.

Wird *x_conlim* = T_NOLIMIT angegeben, so kann der Prozess das bei der Installation von CMX festgelegte Maximum an Verbindungen gleichzeitig halten.

Standardwert bei Angabe von NULL: T_NOLIMIT

Rückgabewert

T_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. Der Prozess ist nicht angemeldet.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_WPARAMETER

Die Angaben in *x_opt* haben ein falsches Format oder enthalten unzulässige Werte.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die beim Aufruf *t_attach* (siehe Abschnitt „t_attach - Anmelden eines Prozesses bei CMX (attach process)“ auf Seite 111) aufgelisteten Fehlerwerte auftreten.

Siehe auch

x_detach(), t_getloc()

9.7.2 x_concf - Verbindung herstellen (connection confirmation)

x_concf() nimmt ein zuvor mit *x_event()* gemeldetes Ereignis X_CONCF von NEABX entgegen. X_CONCF zeigt an, dass die gerufene TS-Anwendung einen Verbindungswunsch des laufenden Prozesses (*x_conrq()*-Aufruf) positiv beantwortet hat.

x_concf() liefert:

– die Benutzerdaten, die die gerufene TS-Anwendung mitgeschickt hat. Die Benutzerdaten müssen vom laufenden Prozess entgegengenommen werden, da sie das NEABV-Protokoll enthalten. Für die Optionsstruktur ist der Wert NULL deshalb unzulässig.

  Das empfangene Protokoll kann dann mit dem Aufruf *x_neavi()* analysiert werden.

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

Liefert *x_concf()* den Wert T_OK zurück, so ist die Verbindung für den laufenden Prozess aufgebaut. Sobald eine Verbindung aufgebaut ist, liegt die Initiative bei der TS-Anwendung (nicht bei CMX). Sie kann:

– Normaldaten und (falls vereinbart) Vorrangdaten senden oder

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

– die Verbindung umlenken oder abbauen.

Liefert NEABX das Ergebnis X_REPEAT zurück, so war der Aufruf zwar erfolgreich, aber es wurden noch nicht alle Benutzerdaten übernommen. Sie müssen *x_concf()* mit gleichen Parametern noch einmal aufrufen, sobald *x_event()* das Ereignis X_REPCCF anzeigt. Erst dann ist die Verbindung vollständig aufgebaut.

```
#include <cmx.h>
#include <neabx.h>
int   x_concf (int *tref,
               struct x_optc1 *x_opt);
```

-> 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.

<> x_opt

Zeiger auf die Struktur *x_optc1*, in der NEABX die Benutzerdaten hinterlegt. *x_opt* muss immer angegeben werden, da das NEABV-Protokoll Bestandteil der Benutzerdaten ist und immer empfangen werden muss.

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[X_NXPWL];  /* Verbindungspasswort */
<-    int     x_prot;             /* Protokoll Datenphase */
    } ;
```

x_optnr

Optionsnummer. Anzugeben ist:

X_OPTC1

wenn in *x_udatal* eine Reserve für das NEABX-Protokoll berücksichtigt wird.

X_OPTC3

wenn bei der Längenangabe in *x_udatal* die Reserve für das NEABX-Protokoll nicht berücksichtigt wird.

x_udatap

Zeiger auf einen Datenbereich. In diesen Bereich trägt NEABX die Benutzerverbindungsnachricht der gerufenen TS-Anwendung ein. Die Benutzerverbindungsnachricht besteht aus dem NEABV-Protokoll (siehe Abschnitt „NEABV-Protokoll“ auf Seite 239).

Die im NEABV-Protokoll enthaltene Benutzernachricht wird im Code des Partners geliefert. Das heutige Transportsystem auf BS2000-Seite liefert keine Benutzerverbindungsnachricht, die mit *x_concf()* zugestellt wird.

x_udatal
: Vor dem Aufruf muss hier die Länge des bereitgestellten Datenbereichs angegeben werden.

  Den Bereich müssen Sie so groß wählen, dass die Benutzerverbindungsnachricht 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 Verbindung wird nicht hergestellt.

  Beim Aufruf trägt NEABX die Länge der empfangenen Benutzerverbindungsnachricht ein.

x_xdata
: liefert die Antwort der gerufenen TS-Anwendung, ob Vorrangdaten benutzt werden können. Die Antwort ist verbindlich.

  Mögliche Werte von *x_xdata*:

  X_YES
  : Die gerufene TS-Anwendung stimmt dem Vorschlag zu, Vorrangdaten auszutauschen.

  X_NO
  : Die Benutzung von Vorrangdaten wird vom Partner abgelehnt.

x_timeout
: Der Inhalt dieses Feldes ist mit NIL belegt.

x_passwd
: Verbindungspasswort. In Anlehnung an die ISO-Norm wird ankommend i. a. kein Verbindungspasswort zugestellt. *x_passwd* ist mit NIL belegt.

  Wurde beim *x_conrq()* dieser Verbindung für den Parameter *x_prot* der Wert X_SPECIAL angegeben, so wird das Verbindungspasswort von NEABX durchgereicht, sofern vorhanden.

x_prot
: bestimmt, ob in der Datenphase mit NEABX-Protokoll gearbeitet werden soll. Bei NEA-Transportsystemen ist dies eine lokale Absprache. *x_prot* enthält den Wert, der in *x_conrq()* gesetzt wurde. Bei einem ISO-Transportsystem enthält *x_prot* die Antwort (Bestätigung bzw. Ablehnung) der Partner-TS-Anwendung auf diesen Vorschlag.

Mögliche Werte von *x_prot* sind:

X_NEABX
: In der Datenphase wird immer ein NEABX-Protokoll gesendet und erwartet.

X_NOBX
: In der Datenphase wird ohne NEABX-Protokoll gearbeitet.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Die Verbindung ist vollständig aufgebaut. Die Datenphase ist erreicht.

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.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_udatal*

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

X_BADPRPI
: Unbekanntes Protokoll-Identifikations-Byte erhalten. Das Protokollelement ist weder ein CONNECT-ATTENTION noch ein CONNECT-Protokollelement. Die Verbindung kann als reine ISO-Transportverbindung betrieben werden. Sie ist ICMX(NEA) jedoch nicht mehr bekannt.

X_BADPVBYTE
: Fehlerhaftes Protokoll-Versions-Byte im empfangenen NEABX-Protokoll erhalten.

X_NOTCNPE
: Es wurde ein CONNECT-Protokollelement erwartet, aber ein anderes empfangen.

X_MAXDAT
: Beim Verbindungsaufbau wurden Benutzerdaten empfangen, die mehr als X_MSG_SIZE byte lang ist.

X_NOINFO
: Keine TIDU-Länge bestimmbar. Die Verbindung wurde wieder abgebaut.

X_NOOPT
: Es wurde kein *x_opt*-Zeiger angegeben.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_concf - Verbindung herstellen (connect confirmation)“ auf Seite 123 aufgelisteten Fehlerwerte und der folgende Fehler auftreten:

T_WSEQUENCE
: Für die in *tref* angegebene Verbindung darf kein *x_concf()* aufgerufen werden.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_conrq(), x_error(), x_event(), x_neavi()

9.7.3 x_conin - Verbindungswunsch entgegennehmen (connection indication)

x_conin() nimmt ein zuvor mit *x_event()* gemeldetes Ereignis X_CONIN entgegen. X_CONIN zeigt an, dass eine rufende TS-Anwendung eine Verbindung zum laufenden Prozess aufbauen will.

Der Aufruf liefert:

– die TRANSPORTADRESSE der rufenden TS-Anwendung,

– den LOKALEN NAMEN der lokalen TS-Anwendung,

– die Benutzerdaten, die die rufende TS-Anwendung beim *x_conrq()* mitgegeben hat. Die Benutzerdaten enthalten das NEABV-Protokoll. Das NEABV-Protokoll kann mit Hilfe des Aufrufs *x_neavi()* analysiert werden.

Anschließend kann der Verbindungswunsch mit *x_conrs()* beantwortet (bestätigt) oder mit *x_disrq()* abgelehnt werden.

Liefert NEABX nach dem Aufruf von *x_conin()* den Wert X_REPEAT zurück, so war der Aufruf zwar erfolgreich, aber NEABX hat noch nicht alle Benutzerdaten empfangen.

x_conin() muss dann mit gleichen Parametern noch einmal aufgerufen werden, sobald *x_event()* das Ereignis X_REPCIN anzeigt.

```
#include <cmx.h>
#include <neabx.h>
int   x_conin (int *tref,
               union x_address *toaddr,
               union x_address *fromaddr,
               struct x_optc1 *x_opt);
```

-> 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.

<- toaddr
: Zeiger auf eine Union *x_address*, in die NEABX den LOKALEN NAMEN der lokalen TS-Anwendung einträgt. Diese Information ist dann wichtig, wenn ein Prozess mehrere TS-Anwendungen steuert. Sie erfahren dadurch, welcher TS-Anwendung die Verbindungsanforderung zuzuordnen ist.

<- fromaddr

Zeiger auf eine Union *x_adress*, in die NEABX die TRANSPORTADRESSE der rufenden TS-Anwendung einträgt.

Mit Hilfe des Aufrufs *t_getname()* (siehe Abschnitt „t_getname - GLOBALEN NAMEN abfragen (get name)“ auf Seite 183) kann zur TRANSPORTADRESSE der GLOBALE NAME der rufenden TS-Anwendung ermittelt werden.

<> x_opt

Zeiger auf die Struktur *x_optc1*.

Mit dieser Struktur können Sie die Information abfragen, die die rufende TS-Anwendung bei der Verbindungsanforderung mitgeliefert hat.

x_opt muss immer angegeben werden, da das NEABV-Protokoll Bestandteil der Benutzerdaten ist und immer empfangen werden muss. 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

wenn in *x_udatal* eine Reserve für das NEABX-Protokoll berücksichtigt wird.

X_OPTC3

bei der Längenangabe in *x_udatal* wird die Reserve für das NEABX-Protokoll nicht berücksichtigt.

x_udatap

Zeiger auf einen Datenbereich. In diesen Bereich trägt NEABX die Benutzerverbindungsnachricht der rufenden TS-Anwendung ein.

Die Benutzerverbindungsnachricht besteht aus dem NEABV-Protokoll (siehe Abschnitt „NEABV-Protokoll“ auf Seite 239).

Das NEABV-Protokoll kann mit dem NEABX-Aufruf *x_neavi()* analysiert werden.

x_udatal
: Vor dem Aufruf geben Sie als Länge des bereitgestellten Datenbereichs *x_udatap* an. 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.

  Beim Aufruf trägt NEABX die Länge der empfangenen Benutzerverbindungsnachricht ein.

x_xdata
: In diesem Feld teilt NEABX mit, ob die rufende TS-Anwendung für diese Verbindung die Benutzung von Vorrangdaten vorgeschlägt.

  Mögliche Werte von *x_xdata*:

  X_YES
  : Der Austausch von Vorrangdaten wird vorgeschlagen.

  X_NO
  : Es wird vorgeschlagen, keine Vorrangdaten auszutauschen.

x_timeout
: Der Inhalt dieses Feldes ist stets X_NO.

x_passwd
: Verbindungspasswort. In Anlehnung an die ISO-Norm wird ankommend i. a. kein Verbindungspasswort zugestellt. *x_passwd* ist dann mit NIL belegt.

  Wird vor dem Aufruf *x_conin()* für den Parameter *x_prot* der Wert X_SPECIAL angegeben, so wird das Verbindungspasswort von NEABX durchgereicht, sofern vorhanden.

x_prot
: Vor dem Aufruf von *x_conin()* können Sie für diesen Parameter den Wert X_SPECIAL angeben. X_SPECIAL bewirkt, dass ein von der rufenden TS-Anwendung gesendetes Verbindungspasswort von NEABX durchgereicht wird.

  Nach dem Aufruf enthält *x_prot* den Vorschlag des Partners, ob in der Datenphase das NEABX-Protokoll ausgetauscht werden soll. Beim NEA-Transportsystem ist dies eine lokale Absprache zwischen der TS-Anwendung und dem CCP. Bei ISO-Transportsystemen ist es eine Absprache zwischen den beiden TS-Anwendungen.

Mögliche Werte:

X_NEABX
: In der Datenphase soll das NEABX-Protokoll bearbeitet werden. (Wird bei NEA-Transportsystemen immer lokal gesetzt.)

X_NOBX
: In der Datenphase soll ohne NEABX-Protokoll gearbeitet werden.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Die Verbindungsanforderung wurde vollständig übernommen.

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.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_udatal*.

X_BADPRPI
: Unbekanntes Protokoll-Identifikations-Byte erhalten. Das Protokollelement ist weder ein CONNECT-ATTENTION noch ein CONNECT-Protokollelement. Die Verbindung kann als reine ISO-Transportverbindung betrieben werden. Sie ist ICMX(NEA) jedoch nicht mehr bekannt.

X_BADPVBYTE
: Fehlerhaftes Protokoll-Versions-Byte im empfangenen NEABX-Protokoll erhalten.

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

X_BADTRANS
: Das Transportsystem, über das die Verbindung aufgebaut werden soll, ist nicht bekannt.

X_MAXDAT
: Beim Verbindungsaufbau wurden Benutzerdaten empfangen, die mehr als X_MSG_SIZE byte lang ist.

X_NOINFO
: Keine TIDU-Länge bestimmbar. Die Verbindung wurde wieder abgebaut.

X_NOOPT
: Es wurde kein *x_opt*-Zeiger angegeben.

X_NOTCNPE
: Es wurde ein CONNECT-Protokollelement erwartet, aber ein anderes empfangen.

X_WPARAMETER
: Die in *x_opt* angegebenen Optionen haben ein falsches Format oder enthalten unzulässige Werte.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_concf - Verbindung herstellen (connect confirmation)“ auf Seite 123 aufgelisteten Fehlerwerte und der folgende Fehler auftreten:

T_WSEQUENCE
: Für die in *tref* angegebene Verbindung darf kein *x_conin()* aufgerufen werden.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_conrq(), x_error(), x_event()

9.7.4 x_conrq - Verbindung anfordern (connection request)

x_conrq() fordert den Aufbau einer Transportverbindung von der lokalen TS-Anwendung zu einer gerufenen TS-Anwendung an (aktiver Verbindungsaufbau). Beim Aufruf muss der laufende Prozess die TRANSPORTADRESSE der gerufenen TS-Anwendung und den LOKALEN NAMEN der rufenden TS-Anwendung übergeben. TRANSPORTADRESSE sowie LOKALEN NAMEN liefert der TNSX als Eigenschaft zum jeweiligen GLOBALEN NAMEN. Sie können vor dem *x_conrq()* mit Hilfe der ICMX(L)-Aufrufe *t_getaddr()* oder *t_getloc()* ermittelt werden.

Die Verbindungsanforderung *x_conrq()* bewirkt im Einzelnen:

– NEABX richtet für die angeforderte Verbindung ein Transport Connection Endpoint (TCEP) ein.

– Die gerufene TS-Anwendung erhält das Ereignis X_CONIN als Verbindungsaufbauanzeige. Sie muss darauf antworten. Die Antwort der gerufenen TS-Anwendung wird dem laufenden Prozess später beim *x_event()* als Ereignis X_CONCF oder X_DISIN von NEABX angezeigt.

– Der gerufenen TS-Anwendung werden mit der Verbindungsanzeige Benutzerdaten in Form des NEABV-Protokolls übermittelt. Das NEABV-Protokoll kann mit Hilfe des ICMX(NEA)-Aufrufs *x_neavo()* erzeugt werden.

Liefert NEABX nach dem Aufruf von *x_conrq()* den Wert X_REPEAT zurück, so hat NEABX noch nicht alle Benutzerdaten an das Transportsystem übergeben. *x_conrq* muss dann mit gleichen Parametern noch einmal aufgerufen werden, sobald *x_event* das Ereignis X_REPCRQ anzeigt.

```
#include <cmx.h>
#include <neabx.h>
int   x_conrq (int *tref,
               union x_address *toaddr,
               union x_address *fromaddr,
               struct x_optc1 *x_opt);
```

<- tref

Zeiger auf die Transportreferenz. Die Transportreferenz wird von NEABX beim ersten Aufruf von *x_conrq()* eingetragen und kennzeichnet diese Verbindung für NEABX eindeutig. Sie ist bei allen Aufrufen anzugeben, die sich auf diese Verbindung beziehen. Insbesondere muss der Inhalt von *tref* bei einem wiederholten Aufruf von *x_conrq()* angegeben werden.

-> toaddr

Zeiger auf eine Union, in der die TRANSPORTADRESSE der gerufenen TS-Anwendung anzugeben ist. *x_address* ist in <*neabx.h*> definiert.

-> fromaddr

Zeiger auf eine Union, in der der LOKALE NAME der rufenden TS-Anwendung anzugeben ist. Abgesehen vom Wiederholungsfall ist derselbe LOKALE NAME anzugeben wie beim Aufruf *x_attach()* für diese TS-Anwendung. *x_address* ist in <*neabx.h*> definiert.

-> x_opt

Zeiger auf die Struktur *x_optc1*. Mit dieser Struktur können Sie der gerufenen TS-Anwendung Informationen mitschicken. Diese erhält die Daten mit der Entgegennahme der Verbindungsanforderung.

x_opt muss angegeben werden, da das NEABV-Protokoll immer gesendet werden muss.

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;
                                        /* Vorrangdatenauswahl */
->          int     x_timeout;          /* Inaktiv-Zeit */
->          char    x_passwd[X_NXPWL];
                                        /* Verbindungspasswort */
->          int     x_prot;             /* Protokoll Datenphase */
    } ;
```

x_optnr

Optionsnummer. Anzugeben ist:

X_OPTC1

wenn bei der Längenangabe in *x_udatal* eine Reserve für das NEABX-Protokoll berücksichtigt wird. X_OPTC1 ist anzugeben, wenn zuvor beim Aufruf von *x_neavo()* zur Erzeugung des NEABV-Protokolls die Optionsnummer X_OPTRK angegeben wurde.

X_OPTC3

bei der Längenangabe in *x_udatal* wird die Reserve für das NEABX-Protokoll nicht berücksichtigt. X_OPTC3 ist anzugeben, wenn zuvor beim Aufruf von *x_neavo()* zur Erzeugung des NEABV-Protokolls die Optionsnummer X_OPTRK1 angegeben wurde.

x_udatap
: Zeiger auf einen Bereich mit Daten, die NEABX an die gerufene TS-Anwendung übergibt.

  Der Datenbereich enthält nur die Benutzerverbindungsnachricht (nicht das NEABX-Protokoll). Die Benutzerverbindungsnachricht besteht aus dem NEABV-Protokoll (siehe Abschnitt „NEABV-Protokoll“ auf Seite 239).

  Das NEABV-Protokoll kann auch mit dem NEABX-Aufruf *x_neavo()* erzeugt und hier direkt übergeben werden.

x_udatal
: Länge des Bereichs *x_udatap*, der von NEABX übergeben werden soll. Die Länge umfasst bei Angabe der Optionsnummer X_OPTC1 die Benutzernachricht plus einer Reserve für das NEABX-Protokoll (8 byte), bei der Angabe der Optionsnummer X_OPTC3 nur die Benutzerverbindungsnachricht (NEABV-Protokoll). Das NEABV-Protokoll kann mit *x_neavo()* erzeugt werden und die Länge, die *x_neavo()* liefert, hier direkt angegeben werden.

  Maximale Länge:

  bei X_OPTC1: X_MSG_SIZE
  bei X_OPTC3: X_MSG_SIZENEU

  Minimale Länge:

  bei X_OPTC1: X_RKMSGMIN + 8 byte

  bei X_OPTC3: X_RKMSGMIN

x_xdata
: In *x_xdata* schlägt der laufende Prozess der gerufenen TS-Anwendung vor, den Austausch von Vorrangdaten zu erlauben, bzw. er schließt ihn aus.

  Mögliche Angaben:

  X_YES
  : Vorrangdaten senden und empfangen wird vorgeschlagen. Bei Rechnerkopplung ist für die Kommunikation mit TIAM-, DCAM-, UTM- und anderen BS2000-Anwendungen immer X_YES anzugeben.

  X_NO
  : Die Benutzung von Vorrangdaten wird ausgeschlossen.

x_timeout

X_NO
keine Zeitüberwachung.

n
Die Verbindung darf n Sekunden lang inaktiv sein. Danach baut NEABX die Verbindung ab. n ist als Dezimalzahl anzugeben.

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. In Anlehnung an die ISO-Norm sollte das Feld mit NIL versorgt werden.

x_prot
In *x_prot* schlägt der laufende Prozess vor, ob in der Datenphase mit NEABX-Protokoll gearbeitet werden soll oder nicht. Bei NEA-Transportsystemen ist dies eine lokale Absprache, d. h. beim Aufruf *x_concf()* wird der Vorschlag immer bestätigt. Bei ISO-Transportsystemen wird der Partner-TS-Anwendung der Vorschlag zugestellt. Diese kann ihn ablehnen oder bestätigen. Die Antwort wird dem laufenden Prozess beim *x_concf()* zugestellt.

Mögliche Angaben:

X_NEABX oder NIL
In der Datenphase soll das NEABX-Protokoll bearbeitet werden.

X_NOBX
In der Datenphase soll nicht mit dem NEABX-Protokoll gearbeitet werden.

Der folgende Wert kann durch | (ODER) verknüpft zusätzlich zu X_NEABX bzw. X_NOBX angegeben werden:

X_SPECIAL
ICMX(NEA) behandelt folgende Punkte in spezieller Form:

– Beim Aufruf *x_concf()* dieser Verbindung wird das Verbindungspasswort der Partner-TS-Anwendung an den laufenden Prozess durchgereicht, sofern eines eintrifft.

– In der Datenphase dieser Verbindung werden Transportquittungsanforderungen nicht von NEABX behandelt, sondern an den laufenden Prozess durchgereicht. Dieser muss dann selbst Transportquittungen senden und kann selbst Transportquittungen anfordern.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Die Verbindungsanforderung wurde vollständig an das Transportsystem übermittelt.

X_REPEAT
: Der Aufruf war erfolgreich. Wenn *x_event()* das Ereignis X_REPCRQ anzeigt, muss *x_conrq()* mit gleichen Parametern nochmals aufgerufen werden.

X_DATASTOP
: Der Aufruf war erfolgreich. Alle Benutzerdaten wurden abgesendet. In einer folgenden Datenphase muss zunächst das Ereignis X_DATAGO abgewartet werden.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_udatal*

X_BADPROT
: *x_prot* enthält keinen der Werte X_NEABX, X_NOBX oder NIL.

X_BADTABLE
: Die beim wiederholten *x_conrq()* angegebene *tref* ist NEABX unbekannt. Sie wurde in der entsprechenden Tabelle nicht gefunden.

X_BADTRANS
: Das Transportsystem, über das die Verbindung aufgebaut werden soll, ist nicht bekannt.

X_NOOPT

Es wurde kein *x_opt*-Zeiger angegeben.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_concf - Verbindung herstellen (connect confirmation)“ auf Seite 123 aufgelisteten Fehlerwerte auftreten.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_attach(), x_error(), x_concf(), x_event(), t_getaddr(), t_getloc

9.7.5 x_conrs - Verbindungsanforderung bestätigen (connection response)

Mit *x_conrs()* akzeptiert (bestätigt) die gerufene TS-Anwendung den Verbindungsaufbauwunsch einer rufenden TS-Anwendung. Der Verbindungsaufbauwunsch wurde dem laufenden Prozess zuvor beim *x_event()* mit dem Ereignis X_CONIN angezeigt. Er muss das Ereignis X_CONIN vor dem *x_conrs()* mit *x_conin()* entgegengenehmen (passiver Verbindungsaufbau). Der rufenden TS-Anwendung wird diese Verbindungsbestätigung als Ereignis X_CONCF zugestellt.

Mit der Antwort *x_conrs()*

– müssen Benutzerdaten an NEABX übergeben werden. Die Benutzerdaten werden in Form des NEABV-Protokolls übergeben. Das NEABV-Protokoll kann mit Hilfe des ICMX(NEA)-Aufrufs *x_neavo()* erzeugt werden.

– ist die Verbindung für den laufenden Prozess fertig aufgebaut.

Nach dem erfolgreichen Aufruf von *x_conrs()* ist die Verbindung hergestellt. Die Initiative liegt jetzt bei der TS-Anwendung. Sie kann:

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

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

– die Verbindung abbauen oder umlenken.

```
#include <cmx.h>
#include <neabx.h>
int   x_conrs (int *tref,
               struct x_optc1 *x_opt);
```

-> 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.

-> x_opt

Zeiger auf die Struktur *x_optc1*. *x_opt* ist immer anzugeben, da das NEABV-Protokoll übergeben werden muss.

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;            /* Vorrangdatenauswahl */
->    int     x_timeout;          /* Inaktiv-Zeit */
->    char    x_passwd[X_NXPWL];  /* Verbindungspasswort */
->    int     x_prot;             /* Protokoll
                                     Datenphase */
    } ;
```

x_optnr

Optionsnummer. Anzugeben ist:

X_OPTC1

wenn in *x_udatal* eine Reserve für das NEABX-Protokoll berücksichtigt wird.

X_OPTC1 ist anzugeben, wenn zuvor beim Aufruf von *x_neavo()* zur Erzeugung des NEABV-Protokolls die Optionsnummer X_OPTRK angegeben wurde.

X_OPTC3

bei der Längenangabe in *x_udatal* wird die Reserve für das NEABX-Protokoll nicht berücksichtigt.

X_OPTC3 ist anzugeben, wenn zuvor beim Aufruf von *x_neavo()* zur Erzeugung des NEABV-Protokolls die Optionsnummer X_OPTRK1 angegeben wurde.

x_udatap

Zeiger auf einen Bereich mit Daten, die NEABX an die rufende TS-Anwendung übergibt. Der Datenbereich enthält nur die Benutzerverbindungsnachricht (nicht das NEABX-Protokoll). Die Benutzerverbindungsnachricht besteht aus dem NEABV-Protokoll (siehe Abschnitt „NEABV-Protokoll“ auf Seite 239). Das NEABV-Protokoll kann auch mit dem NEABX-Aufruf *x_neavo()* erzeugt und direkt übergeben werden.

x_udatal

Länge des Bereichs *x_udatap*, der von NEABX übergeben werden soll. Die Länge umfasst bei Angabe der Optionsnummer X_OPTC1 die Benutzerverbindungsnachricht plus einer Reserve

für das NEABX-Protokoll (8 byte), bei der Angabe der Optionsnummer X_OPTC3 nur die Benutzerverbindungsnachricht (NEABV-Protokoll). Wird das NEABV-Protokoll mit *x_neavo()* erzeugt, so kann die Länge, die *x_neavo()* liefert, hier direkt angegeben werden.

Maximale Länge:

bei X_OPTC1: X_MSG_SIZE
bei X_OPTC3: X_MSG_SIZENEU

Minimale Länge:

bei X_OPTC1: X_RKMSGMIN + 8 byte

bei X_OPTC3: X_RKMSGMIN

x_xdata

In *x_xdata* antwortet der laufende Prozess 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.

Mögliche Angaben:

X_YES

Vorschlag für Vorrangdaten senden und empfangen wird akzeptiert.

Für die Kommunikation mit TIAM-, DCAM-, UTM-Anwendungen im BS2000/OSD ist i.a. X_YES anzugeben.

X_NO

Benutzung von Vorrangdaten wird abgelehnt.

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

Antwort auf den Vorschlag, ob in der Datenphase mit dem NEABX-Protokoll gearbeitet werden soll oder nicht. Bei NEA-Transportsystemen ist dies immer eine lokale Abmachung. Bei ISO-Transportsystemen wird der Austausch des NEABX-Protokolls mit der Partner-TS-Anwendung ausgehandelt.

Mögliche Angaben:

X_NEABX oder NIL

Der Vorschlag in der Datenphase das NEABX-Protokoll zu bearbeiten, wird angenommen. Bei Verbindungen zu DCAM-, TIAM-, UTM-Anwendungen ist immer X_NEABX anzugeben.

X_NOBX

In der Datenphase wird ohne NEABX-Protokoll gearbeitet. Der folgende Wert kann, durch | (ODER) verknüpft, zusätzlich zu X_NEABX bzw. X_NOBX angegeben werden:

X_SPECIAL

In der Datenphase dieser Verbindung werden Transportquittungsanforderungen nicht von NEABX behandelt, sondern an den laufenden Prozess durchgereicht. Dieser muss dann selbst Transportquittungen senden und kann selbst Transportquittungen anfordern.

Rückgabewert

T_OK

Der Aufruf war erfolgreich. Die Verbindung ist vollständig aufgebaut.

X_DATASTOP

Der Aufruf war erfolgreich. Alle Benutzerdaten wurden abgesendet. In einer folgenden Datenphase muss zunächst das Ereignis X_DATAGO abgewartet werden.

X_ERROR

Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_udatal*

X_BADPROT
: *x_prot* enthält keinen der Werte X_NEABX, X_NOBX oder NIL.

X_BADTABLE
: Die angegebene *tref* ist NEABX unbekannt. Es wurde in der entsprechenden Tabelle nicht gefunden.

X_NOINFO
: Es ist keine TIDU-Länge bestimmbar. Die Verbindung wurde wieder abgebaut.

X_NOOPT
: Es wurde kein *x_opt*-Zeiger angegeben.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_concf - Verbindung herstellen (connect confirmation)“ auf Seite 123 aufgelisteten Fehlerwerte und der folgende Fehler auftreten:

T_WSEQUENCE
: Für die in *tref* angegebene Verbindung darf kein *x_conrs()* aufgerufen werden.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_conin(), x_error(), x_event()

9.7.6 x_datago - Datenfluss freigeben (datago)

x_datago() gibt die gesperrte Datenanzeige auf der angegebenen Verbindung frei. Der laufende Prozess teilt NEABX dadurch mit, dass er wieder bereit ist, Daten entgegenzunehmen. Dieser Aufruf gibt auch die Vorrangdatenanzeige (sofern vereinbart) wieder frei, falls sie (auch) gesperrt war. Der Aufruf bewirkt im Einzelnen:

– der laufende Prozess erhält wieder die Ereignisse X_DATAIN und X_XDATIN für die angegebene Verbindung zugestellt, falls sie anstehen,

– die sendende TS-Anwendung erhält (im Verlauf) das Ereignis X_DATAGO zugestellt, sie darf wieder Daten senden.

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

-> tref
: Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, für die Sie den Datenfluss freigeben wollen.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Die Datensperre ist aufgehoben.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden. Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADTABLE
: Die angegebene *tref* ist nicht in der Tabelle der Verbindungen enthalten, die NEABX bekannt sind. Entweder ist sie keiner Verbindung zugeordnet, oder die zugehörige Verbindung ist nicht über ICMX(NEA) aufgebaut worden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_concf - Verbindung herstellen (connect confirmation)“ auf Seite 123 aufgelisteten Fehlerwerte und der folgende Fehler auftreten:

T_WSEQUENCE
: Die in *tref* angegebene Verbindung ist noch nicht vollständig aufgebaut.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_datastop(), x_xdatstop(), x_error(), x_event()

9.7.7 x_datain - Daten empfangen (data indication)

Mit *x_datain()* nimmt der laufende Prozess ein zuvor mit *x_event()* gemeldetes Ereignis X_DATAIN entgegen. Er übernimmt damit auf der angegebenen Verbindung eine Dateneinheit (TIDU), die zu der aktuell übertragenen Nachricht (TSDU) der sendenden TS-Anwendung gehört.

Der Indikator *x_chain* zeigt an, ob noch eine weitere TIDU zur TSDU gehört oder nicht. Jede weitere TIDU zeigt NEABX erneut mit dem Ereignis X_DATAIN an.

Die Länge einer TIDU ist abhängig vom verwendeten Transportsystem. Sie kann für eine bereits aufgebaute Verbindung mit *x_info()* ermittelt werden. Keine TIDU muss vollständig gefüllt sein. Die Aufteilung einer TIDU ist rein lokal und erlaubt keine Rückschlüsse auf die Aufteilung einer TSDU in TIDUs bei der sendenden TS-Anwendung.

Eine mit *x_datain()* empfangene TIDU kann kürzer oder länger als die mit *x_datarq()* gesendete TIDU sein. Ist sie kürzer, dann steht im Indikator *x_chain* X_MORE und *x_event()* zeigt mit dem Ereignis X_DATAIN weitere zum Empfang bereitstehende Daten an.

Wenn Sie nicht bereit sind, Daten zu empfangen, können Sie mit *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.

Wurde beim Aufbau der Verbindung die Behandlung von Transportquittungen in der TS-Anwendung vereinbart, so bedeutet die Rückgabe des Wertes X_ERROR mit Ergebnis X_QUITPE beim Aufruf von *x_error()*, dass für den laufenden Prozess eine Transportquittung eingetroffen ist. Die Optionselemente *x_quit* und *x_seqno* sind dann entsprechend versorgt.

```
#include <cmx.h>
#include <neabx.h>
int   x_datain (int *tref,
                char *x_datap,
                int *x_datal,
                int *x_chain,
                x_optd *x_opt);
```

-> 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.

<- x_datap

Zeiger auf einen Bereich, in den NEABX die empfangenen Daten einträgt.

Ist *x_opt* gleich NULL, so übergibt NEABX alle empfangenen Daten an die lokale TS-Anwendung.

Ist *x_opt* ungleich NULL, so wird das NEABX-Protokoll von NEABX vor der Übergabe an die lokale TS-Anwendung der angegebenen Optionsnummer entsprechend behandelt:

X_OPTD1:

Der Speicherbereich muss eine Reserve für das NEABX-Protokoll enthalten (kompatibler Modus zu CMX V2.1).

X_OPTD2:

Das NEABX-Protokoll wird für die TS-Anwendung nicht sichtbar.

X_OPTD3:

Das NEABX-Protokoll wird an den Beginn des Datenbereiches gestellt und im Element *x_offset* der Option *x_optd3* wird die Länge des Protokolles hinterlegt, und somit der TS-Anwendung mitgeteilt, wo die Nettonachricht beginnt.

<> 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. Die zurückgelieferte Länge betrifft immer nur die Nettodatenlänge auch bei Verwendung der Option X_OPTD3.

<- x_chain

Zeiger auf einen Indikator, mit dem NEABX anzeigt, ob noch weitere TIDUs zur TSDU gehören.

Folgende Werte sind möglich:

X_MORE

Es folgt noch mindestens eine weitere TIDU, die zu der TSDU gehört. Für jede weitere TIDU meldet NEABX ein eigenes Ereignis X_DATAIN.

X_END

Es ist keine weitere TIDU vorhanden. Die TSDU ist fertig übertragen.

<> x_opt

Zeiger auf eine Union *x_optd*, die eine der Strukturen *x_optd1*, *x_optd3* oder die Angabe NULL enthält. Die Angabe von *x_opt* ist obligatorisch, wenn für die Verbindung vereinbart wurde, dass in der Datenphase mit NEABX-Protokoll gearbeitet werden soll, und die erste TIDU einer TSDU empfangen wird. Die Angabe von NULL ist obligatorisch, wenn

a) eine weitere TIDU einer TSDU entgegengenommen werden soll, also die vorhergehende TIDU 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 Strukturen *x_optd1* und *x_optd3* und die Union *x_optd* sind in der Datei <*neabx.h*> definiert.

```
    struct x_optd1 {
->     int   x_optnr;   /* Optionsnummer,X_OPTD1,X_OPTD2 */
<-     int   x_code;    /* Nachrichtencode */
<-     int   x_strukt;  /* Nachrichtenstruktur */
<-     int   x_quit;    /* Transportquittungen */
<-     short x_seqno;   /* Nachrichtensequenznummer */
    };

    struct x_optd3 {
->     int   x_optnr;   /* Optionsnummer,X_OPTD3 */
<-     int   x_code;    /* Nachrichtencode */
<-     int   x_strukt;  /* Nachrichtenstruktur */
<-     int   x_quit;    /* Transportquittungen */
<-     short x_seqno;   /* Nachrichtensequenznummer */
<-     int   x_offset;  /* Offset bis zum Beginn der
                           Daten */
    };
```

x_optnr

Optionsnummer. Mögliche Angaben:

X_OPTD1 oder X_OPTD2 bei *x_optd1*

X_OPTD3 bei *x_optd3*

Die Bedeutung der Werte ist bei *x_datap* beschrieben.

x_code

bezeichnet den Nachrichtencode. Es bedeutet:

X_ASCII

Die eingegangenen Daten sind in ASCII codiert.

X_EBCDIC

Die eingegangenen Daten sind in EBCDIC codiert.

X_TRANS (= X_EBCIDC)
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.

x_quit
ist nur relevant, wenn beim Verbindungsaufbau die Behandlung von Transportquittungen in der TS-Anwendung angegeben wurde.

Wurde ein DATA-Protokollelement empfangen, sind folgende Werte für *x_quit* möglich:

0 keine Quittung verlangt.

1 Transportquittung wird verlangt.

Wurde ein Quittungsprotokollelement empfangen (Rückmeldung X_ERROR mit Fehler X_QUITPE), sind folgende Werte für *x_quit* möglich:

1 positive Quittung erhalten.

2 negative Quittung erhalten.

x_seqno
: enthält die Nachrichtensequenznummer, sofern x_quit nicht NULL ist.

x_offset
: In dieses Feld liefert NEABX die Länge des NEABX-Protokolls zurück. Der TS-Anwendung wird so mitgeteilt, wo die Nettonachricht beginnt. *x_offset* gibt die Distanz von *x_datap* bis zum Nettodatenbeginn an.

Rückgabewert

T_OK
: Die Dateneinheit ist vollständig gelesen.

X_DATASTOP
: Die Daten wurden vollständig vom Transportsystem übernommen, aber beim Senden einer im NEABX-Protokoll verlangten Transportquittung wurde die Datensendesperre angezeigt. Es muss das Ereignis X_DATAGO abgewartet werden.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_datal*

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

X_NOTDTPE
: DATA-Protokollelement erwartet, aber nicht erhalten.

X_QUITPE
: *x_datain()* hat ein QUITTUNG-Protokollelement erhalten. Die Optionselemente *x_quit* und *x_seqno* sind entsprechend versorgt.

X_BADDTPELI
: Die im empfangen NEABX-Protokoll angegebene Länge des DATA-Protokollelements stimmt nicht.

X_WPARAMETER
: Fehlerhafter Parameter, es wurde ein falscher Wert in *x_optnr* angegeben.

X_WXOPT
: Falsche *x_opt* Angabe:

  x_opt != NULL, obwohl kein NEABX-Protokoll;

  x_opt != NULL, obwohl zweite und folgende TIDU empfangen

  x_opt = NULL, obwohl NEABX-Protokoll vereinbart und erste TIDU der TSDU empfangen.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_datain - Daten empfangen (data indication)“ auf Seite 141 und Abschnitt „t_vdatain - Daten empfangen (data indication)“ auf Seite 203 aufgelisteten Fehlerwerte und der folgende Fehlerwert auftreten:

T_WSEQUENCE
: Die in *tref* angegebene Verbindung ist noch nicht vollständig aufgebaut.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_error(), x_event(), x_info()

9.7.8 x_datarq - Daten senden (data request)

Mit *x_datarq()* senden Sie die nächste Transport Interface Data Unit (TIDU) einer Transport Service Data Unit (TSDU) 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.

```
#include <cmx.h>
#include <neabx.h>
int   x_datarq (int *tref,
                char *x_datap,
                int *x_datal,
                int *x_chain,
                x_optd *x_opt);
```

-> tref
: Zeiger auf die Transportreferenz. Hier geben Sie die Transportreferenz der Verbindung an, auf der Sie Daten senden wollen.

-> x_datap
: Zeiger auf einen Bereich, in dem die Daten stehen, die Sie senden wollen. Ist *x_opt* ungleich NULL wird der Optionsnummer entsprechend die angegebene Länge der Daten entgegengenommen. In jedem Fall muss die TS-Anwendung nur Angaben über die Nettodaten machen. Über eventuell vorgesetzte Protokolle ist der TS-Anwendung nichts bekannt.

-> x_datal
: Zeiger auf die Längenangabe **x_datal*. Bei der Angabe der Länge **x_datal* ist Folgendes zu beachten:

– wurde beim Verbindungsaufbau vereinbart, dass während der Datenphase kein NEABX-Protokoll ausgetauscht wird (*x_opt* = NULL), so entspricht **x_datal* genau der Länge der zu sendenden Daten aus *x_datap*. Es gilt:

  Maximalwert für **x_datal* = *x_maxl* - X_DRQPHL

  Minimalwert für **x_datal* = 1 byte (1 byte Sendedaten)

  x_maxl ist die TIDU-Länge (maximale Länge einer Dateneinheit). Sie ist mit *x_info()* zu ermitteln.

– wurde beim Verbindungsaufbau der Austausch des NEABX-Protokolls während der Datenphase vereinbart, so ist die anzugebende Datenlänge abhängig von der in *x_opt* angegebenen Optionsnummer *x_optnr*. Die folgenden Angaben sind einzuhalten:

  x_optnr = X_OPTD1
  **x_datal* muss um X_DRQPHL größer angegeben werden, als es den zu sendenden Nettodaten entspricht. Der Speicherbereich der Anwendung wird jedoch nicht für den Aufbau des Protokolles herangezogen (kompatibler Modus zu CMX V2.1).

  Maximalwert für **x_datal* = *x_maxl*.

  Minimalwert für **x_datal* = X_DRQPHL.

  x_optnr = X_OPTD2
  **x_datal* enthält nur die Nettodatenlänge.

  Der Speicherbereich der TS-Anwendung muss keinerlei Reserven für das NEABX-Protokoll berücksichtigen.

  Maximalwert für **x_datal* = *x_maxl* - X_DRQPHL.

  Minimalwert für **x_datal* = 0 byte

  x_optnr = X_OPTD3
  **x_datal* enthält nur die Nettodatenlänge. Im Element *x_offset* der Optionsstruktur hinterlegt die TS-Anwendung die Distanz von *x_datap* weggerechnet, ab der die Nettodaten beginnen. Diese Distanz muss X_DRQPHL sein.

  Maximalwert für **x_datal* = *x_maxl* - X_DRQPHL.

  Minimalwert für **x_datal* = 0 byte

x_opt = NULL

**x_datal* enthält nur die Nettodatenlänge. Der Speicherbereich der TS-Anwendung muss keinerlei Reserven für das NEABX-Protokoll berücksichtigen.

Maximalwert für **x_datal* = *x_maxl* - X_DRQPHL.

Minimalwert für **x_datal* = 1 byte

x_max ist die TIDU-Länge (maximale Länge einer Dateneinheit). Sie ist mit *x_info()* zu ermitteln.

-> 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 *x_datarq()* erneut aufzurufen.

X_END

Es sind keine weiteren Dateneinheiten vorhanden. Die Nachricht ist fertig übertragen.

-> x_opt

Zeiger auf eine Union *x_optd*, die eine der Strukturen *x_optd1*, *x_optd3* oder die Angabe NULL enthält.

Die Angabe von *x_opt* ist obligatorisch, wenn für die Verbindung vereinbart wurde, dass in der Datenphase mit NEABX-Protokoll gearbeitet werden soll, und die erste TIDU einer TSDU gesendet wird.

Die Angabe NULL ist obligatorisch und nur dann erlaubt, 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 Strukturen *x_optd1* und *x_optd3* und die Union *x_optd* sind in der Datei <*neabx.h*> definiert.

```
    struct x_optd1 {
->      int   x_optnr;
                              /* Optionsnummer,X_OPTD1,X_OPTD2 */
->      int   x_code;         /* Nachrichtencode */
->      int   x_strukt;       /* Nachrichtenstruktur */
->      int   x_quit;         /* Transportquittungen */
->      short x_seqno;
                              /* Nachrichtensequenznummer */
    };

    struct x_optd3 {
->      int   x_optnr;        /* Optionsnummer,X_OPTD3 */
->      int   x_code;         /* Nachrichtencode */
->      int   x_strukt;       /* Nachrichtenstruktur */
->      int   x_quit;         /* Transportquittungen */
->      short x_seqno;
                              /* Nachrichtensequenznummer */
->      int   x_offset;
                              /* Offset bis zum Beginn der Daten */
    };
```

x_optnr

Optionsnummer. Mögliche Angaben:

X_OPTD1 oder X_OPTD2 bei *x_optd1*

X_OPTD3 bei *x_optd3*

Die Bedeutung der Werte ist bei *x_datal* beschrieben.

x_code

bezeichnet den Nachrichtencode der Daten in *x_datap*:

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. Die Daten müssen in dem Code vorliegen, den der Partner erwartet.

Bei ISO- oder NEA-Anschluss muss ein mitgeliefertes Benutzerdienstprotokoll in dem Code vorliegen, den der Partner erwartet. Die Benutzerdienstprotokolle sind somit transparent.

x_strukt
: Nachrichtenstruktur.

  Es ist X_ETXEOT anzugeben.

x_quit
: ist nur relevant, wenn beim Verbindungsaufbau die Behandlung von Transportquittungen in der TS-Anwendung angegeben wurde.

  Falls Daten gesendet werden sollen (*x_datal* != 0), wird in das NEABX-Protokoll das Quittungsanforderungsbit QVBIT gesetzt.

  Falls keine Nutzdaten gesendet werden sollen (*x_datal* = 0 oder kleiner), sind folgende Werte für *x_quit* möglich:

  0 Fehler

  1 positive Transportquittung wird gesendet.

  2 negative Transportquittung wird gesendet.

x_seqno
: enthält die Nachrichtensequenznummer, sofern x_quit nicht NULL ist.

x_offset
: Im Element *x_offset* hinterlegt die TS-Anwendung die Distanz von *x_datap* weggerechnet, ab der die Nettodaten beginnen. Diese Distanz muss X_DRQPHL sein.

Rückgabewert

T_OK
: Aufruf erfolgreich.

X_DATASTOP
: Der Aufruf war erfolgreich. Sie dürfen aber erst weiter senden, wenn das Ereignis X_DATAGO eingetroffen ist.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_datal*

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

X_BADXCODE
: Der Wert in *x_code* ist falsch.

X_WPARAMETER
: Fehlerhafter Parameter, es wurde ein falscher Wert für *x_optnr* angegeben.

X_WXOPT
: Falsche *x_opt* Angabe:

  x_opt != NULL, obwohl kein NEABX-Protokoll;

  x_opt != NULL, obwohl zweite und folgende TIDU gesendet

  x_opt = NULL, obwohl NEABX-Protokoll vereinbart und erste TIDU der TSDU gesendet.

X_BADSTRUKT
: In *x_strukt* wurde kein erlaubter Wert angegeben.

  Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_datarq - Daten senden (data request)“ auf Seite 144 und Abschnitt „t_vdatarq - Daten senden (data request)“ auf Seite 206 aufgelisteten Fehlerwerte und der folgende Fehlerwert auftreten.

T_WSEQUENCE
: Die in *tref* angegebene Verbindung ist noch nicht vollständig aufgebaut.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_datastop(), x_error(), x_event(), x_info(), x_xdatstop()

9.7.9 x_datastop - Datenfluss stoppen (datastop)

x_datastop() sperrt die Datenanzeige auf der angegebenen Verbindung.

x_datastop() bewirkt im Einzelnen:

– Der laufende Prozess teilt CMX so mit, dass er bis auf weiteres nicht bereit ist, für diese Verbindung Daten zu empfangen. Ein bereits angezeigtes Ereignis X_DATAIN muss aber erst mit *x_datain()* entgegengenommen werden.

– Der laufende Prozess bekommt das Ereignis X_DATAIN für die angegebene Verbindung nicht mehr zugestellt. Er kann aber, während die Datenanzeige gesperrt ist, andere CMX-Funktionen aufrufen, z. B. eine weitere Verbindung aufbauen, abbauen, umlenken. Er kann auch auf der angegebenen Verbindung selber Daten senden, sofern für ihn keine Datensendesperre gesetzt wurde (X_DATASTOP).

– Die sendende TS-Anwendung erhält (im Verlauf) bei *x_datarq()* das Ergebnis X_DATASTOP. Sie darf keine Daten mehr senden (siehe auch Abschnitt „Transportsystem-spezifische Besonderheiten“ auf Seite 104).

Freigegeben wird die Datenanzeige mit *x_datago()*.

Vorrangdaten sind von *x_datastop()* nicht betroffen.

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

-> tref
: Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, für die Sie den Datenfluss stoppen wollen.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_datastop - Datenanzeige sperren (data stop)“ auf Seite 147 aufgelisteten Fehlerwerte und der folgende Fehlerwert auftreten.

T_WSEQUENCE
: Die in *tref* angegebene Verbindung ist noch nicht vollständig aufgebaut.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

9.7.10 x_detach - 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 NEABX sie implizit ab. Im Normalfall sollten jedoch alle Verbindungen vor dem Aufruf von *x_detach()* mit *x_disrq()* abgebaut werden. Sobald Sie den letzten Prozess einer TS-Anwendung abgemeldet haben, ist die TS-Anwendung bei NEABX nicht mehr bekannt. Verbindungsanforderungen für diese TS-Anwendung werden dann nicht mehr angenommen.

```
#include <cmx.h>
#include <neabx.h>
int   x_detach (struct x_myname *name,
                struct x_opta1 *x_opt);
```

-> name
: Zeiger auf die Struktur *x_myname*, in der der LOKALE NAME der TS-Anwendung anzugeben ist. Es wird derselbe LOKALE NAME angegeben wie beim Aufruf *x_attach()*.

-> opt
: Muss NULL gesetzt werden.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_detach - Abmelden Prozess aus TS-Anwendung (detach process)“ auf Seite 149 aufgelisteten Fehlerwerte auftreten.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_attach(), x_error()

9.7.11 x_disin - 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.

x_disin() liefert ferner:

– die Benutzerdaten, die die ferne TS-Anwendung mitgeschickt hat, falls die ferne TS-Anwendung den Verbindungsabbau ausgelöst hat und sofern das Transportsystem diese Option bietet.

– den Grund für den Abbau der Transportverbindung, falls das Ereignis X_DISIN von NEABX oder vom Transportsystem ausgelöst wurde.

  Der Grund für den Verbindungsabbau wird von *x_disin()* in sedezimaler Darstellung geliefert. Die Klartextdarstellung des Codes erhält man mit Hilfe der ICMX(L)-Funktionen *t_preason()* bzw. *t_strreason()*.

```
#include <cmx.h>
#include <neabx.h>
int   x_disin (int *tref,
               int *reason,
               struct x_optc2 *x_opt);
```

-> 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.

<- reason
: Zeiger auf die Ursache des Verbindungsabbaus. Es wird entweder T_USER (der Kommunikationspartner hat die Verbindung abgebaut) oder der Abbaugrund von CMX oder den CCPs ausgegeben, sofern CMX die Verbindung abgebaut hat. Die von CMX oder den CCPs gelieferten Werte sind im Anhang aufgelistet.

<> x_opt

Zeiger auf die Struktur *x_optc2*. 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.

Derzeit können von den möglichen Partner-TS-Anwendungen noch keine Informationen mitgeliefert werden, da die Partner-Transportsysteme keine Schnittstelle zur Übertragung von Benutzerdaten besitzen.

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 Datenpuffers */
    } ;
```

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.

x_udatal

Vor dem Aufruf geben Sie die Länge des bereitgestellten Datenbereichs *x_udatap* an. Der Bereich muss so groß sein, dass die empfangenen Benutzerdaten hineinpassen. Die maximal zulässige Benutzerdatenlänge ist abhängig vom verwendeten Transportsystem. X_MSG_SIZE ist eine für alle Transportsysteme geeignete Maximalgröße. Nach dem Aufruf enthält *x_udatal* die Länge der empfangenen Benutzerdaten.

Rückgabewert

T_OK

Der Aufruf war erfolgreich.

X_ERROR

Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_disin - Verbindungsabbau entgegennehmen (disconnection indication)“ auf Seite 151 aufgelisteten Fehlerwerte auftreten. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_detach(), x_disrq(), x_event()

9.7.12 x_disrq - Verbindung abbauen (disconnection request)

Mit *x_disrq()* können Sie:

– eine bestehende Verbindung abbauen oder

– die Verbindungsanforderung einer fernen TS-Anwendung ablehnen.

Der fernen TS-Anwendung wird in beiden Fällen eine Verbindungsabbauanzeige (X_DISIN) mit Verbindungsabbaugrund T_USER zugestellt.

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 zuvor gesendete Daten 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 zuletzt gesendete TIDU erhalten haben.

Beim Verbindungsabbau können keine Benutzerdaten für die ferne TS-Anwendung übergeben werden, da DCAM keine Schnittstelle bietet, an der der Anwendung Benutzerdaten übergeben werden können.

```
#include <cmx.h>
#include <neabx.h>
int   x_disrq (int *tref,
               struct x_optc2 *x_opt);
```

-> tref

Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, die Sie abbauen wollen. Wenn Sie eine Verbindungsanforderung ablehnen wollen, die von *x_event()* mit dem Ereignis X_CONIN angezeigt wird, so liefert *x_event()* auch die Transportreferenz der abzuweisenden Verbindung.

-> x_opt

Für *x_opt* ist der NULL-Zeiger anzugeben.

TS-Anwendungen im BS2000/OSD über BCAM können keine Benutzerdaten übergeben werden.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_WXOPT
: Falsche *x_opt* Angabe: Für *x_opt* **muss** NULL angegeben werden.

  Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_disrq - Verbindung abbauen (disconnection request)“ auf Seite 154 aufgelisteten Fehlerwerte auftreten.

  Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_conin(), x_disin(), x_event(), x_error()

9.7.13 x_error - Fehlercodes abfragen (error)

x_error() liefert Diagnoseinformationen, wenn ein anderer NEABX-Aufruf das Ergebnis X_ERROR hatte.

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 in NEABX entstandenen Fehlermeldungen werden in sedezimaler Darstellung von *x_error()* übergeben. Fehlermeldungen mit Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können mit Hilfe der Aufrufe *x_strerror()* bzw. *x_perror()* in Klartext übersetzt werden. *x_strerror()* liefert einen Zeiger auf einen statischen Bereich, der die Klartextdarstellung der Fehlermeldung erhält.

x_perror() schreibt den Klartext der Fehlermeldung auf die Standardfehlerausgabe *stderr*.

Der sedezimale Fehlercode kann auch mit Hilfe des Kommandos *cmxdec* (siehe Handbücher „CMX, Betrieb und Administration“ [1] und [2]) decodiert werden.

Der Aufbau der Fehlermeldungen ist in Abschnitt „Fehlerbehandlung“ auf Seite 40 beschrieben.

```
#include <cmx.h>
#include <neabx.h>
int   x_error (void);
```

Rückgabewert

Rückgabewert von *x_error()* ist der hexadezimale Code der von NEABX erzeugten Fehlermeldung (Fehlertyp, Fehlerklasse, Fehlerwert). Die Fehlermeldungen sind in *<neabx.h>* definiert. Eine Liste aller möglichen Fehlerwerte mit Fehlertyp X_BX3 (9) und Fehlerklasse X_NEAERR (B), d. h. alle möglichen Rückgabewerte von *x_error()* finden Sie im Anhang.

Bei der Beschreibung der einzelnen Funktionsaufrufe an ICMX(NEA) sind unter der Überschrift ’Fehler’ jeweils die Fehlerwerte aufgelistet, die *x_error()* im Fall eines Abbruchs der jeweiligen Funktion zurückliefert.

Siehe auch

x_perror(), x_strerror()

9.7.14 x_event - Ereignis abwarten oder abfragen (event)

x_event() stellt fest, ob ein NEABX-Ereignis für den laufenden Prozess eingetroffen ist.

Über den Parameter *x_cmode* kann man den Verarbeitungsmodus von *x_event()* festlegen. *x_event()* kann:

– **synchron** darauf warten, dass ein NEABX-Ereignis für den laufenden Prozess eintrifft. Der Prozess wird während dieser Wartezeit suspendiert.

  Der Wartezustand kann durch Signale unterbrochen werden.

  In den Optionen *x_opt* kann man eine Zeitschranke für das synchrone Warten angeben. Ist innerhalb dieser Wartezeit kein Ereignis eingetroffen, wird der Wartezustand abgebrochen.

– **asynchron** prüfen, ob ein NEABX-Ereignis für den laufenden Prozess vorliegt. Die Funktion kehrt immer sofort zum laufenden Prozess zurück.

Neben dem entsprechenden Ereignis liefert *x_event()*:

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

– ereignisspezifische Zusatzinformationen, falls diese in den Optionen *x_opt* vereinbart wurden.

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

Ausnahmen:

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

– Das Ereignis X_DISIN (Verbindungsabbau-Anzeige) kann die Ereignisse X_DATAIN und X_XDATIN für die betroffene Verbindung überholen und damit zerstören.

  Die Daten, die X_DATAIN bzw. X_XDATIN anzeigen sollte, gehen verloren.

Hinweis

x_event() erlaubt es einer TS-Anwendung innerhalb eines Prozesses, sowohl NEA-konforme als auch ISO-konforme Verbindungen zu unterhalten. D. h. der Prozess nutzt sowohl ICMX(NEA) als auch ICMX(L). Ein solcher Prozess muss selbst unterscheiden, ob eine Transportreferenz zu einer NEA- oder zu einer ISO-Verbindung gehört. *x_event()* meldet auch Ereignisse zu Transportreferenzen, die NEABX nicht bekannt sind.

```
#include <cmx.h>
#include <neabx.h>
int   x_event (int *tref,
               int x_cmode,
               struct x_opte1 *x_opt);
```

<- 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.

-> x_cmode

gibt an, ob *x_event()* synchron auf ein Ereignis warten soll, oder asynchron prüfen soll, ob ein Ereignis vorliegt. Mögliche Werte:

X_WAIT (synchrone Verarbeitung)

Der laufende Prozess wird suspendiert, bis ein Ereignis eintritt, die vereinbarte Wartezeit abläuft (Parameter *x_timeout* in *x_opt*), oder ein Signal ihn weckt. In den letzten beiden Fällen wird das Ereignis X_NOEVENT geliefert. Um den Prozess zu wecken, können Sie alle Signale (*alarm()*) außer SIGTERM verwenden.

X_ CHECK (asynchrone Verarbeitung)

x_event() prüft, ob ein Ereignis ansteht. Wenn nicht, dann kehrt *x_event()* mit X_NOEVENT zurück.

<> x_opt

Für *x_opt* ist der Zeiger auf die Struktur *x_opte1* mit Benutzeroptionen oder der Wert NULL anzugeben.

Die Struktur *x_opte1* ist in der Datei <*neabx.h*> definiert.

```
    struct x_opte1 {
->    int x_optnr ;  /* Optionsnummer */
->    int x_timout;  /* Zeitschranke für X_WAIT */
<-    int x_evdat ;
                     /* ereignisspezifische Zusatzinformationen */
    };
```

x_optnr

Anzugeben ist X_OPTE1.

x_timeout

Für *x_timeout* kann eine Wartezeit in Sekunden angegeben werden. Bei *x_cmode* = X_WAIT unterbricht *x_event()* das synchrone Warten nach Ablauf dieser Wartezeit. Eine Angabe kleiner Null (-1) bedeutet, dass kein Timer aktiviert wird.

Bei *x_cmode* = X_CHECK wird die Angabe für *x_timeout* ignoriert.

x_evdat

Bei den Ereignissen X_DATAIN und X_XDATIN wird in *x_evdat* die Länge der angezeigten Daten angegeben. Diese Länge kann dann bei den Funktionen *x_datain()* und *x_xdatin()* angegeben werden.

Rückgabewert

X_NOEVENT

falls x_cmode = X_CHECK: kein Ereignis da.

falls x_cmode = X_WAIT: Abbruch, z. B. durch ein Signal

oder

T_DATAGO wurde intern von CMX angezeigt, aber beim Absenden einer noch ausstehenden Transportquittung wurde wieder T_DATASTOP angezeigt. Somit ist T_DATAGO aufgehoben und kein eigentliches Ereignis zu melden.

Der Inhalt von *tref* ist nicht definiert.

X_DATAIN
: Auf der in *tref* angegeben Verbindung wurden Daten empfangen.

  Von NEABX erwartete Reaktion: Aufruf von *x_datain()*.

  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 lokale TS-Anwendung kann auf der in *tref* angegebenen Verbindung weiter Daten senden.

  Mögliche Reaktion: *x_datarq()*.

  Das Ereignis X_DATAGO erlaubt es der lokalen TS-Anwendung auch, auf dieser Verbindung wieder Vorrangdaten zu senden, sofern beim Verbindungsaufbau die Benutzung von Vorrangdaten vereinbart wurde.

X_XDATIN
: Auf der in *tref* angegebenen Verbindung wurden Vorrangdaten empfangen. Von NEABX erwartete Reaktion: *x_xdatin()*.

  NEABX zeigt dieses Ereignis nur an, wenn beim Verbindungsaufbau die Benutzung von Vorrangdaten vereinbart wurde.

  Solange der Vorrangdatenfluss gestoppt ist, der empfangende Prozess also für diese Verbindung *x_xdatstop()* gegeben hat, wird dieses Ereignis nicht angezeigt.

X_XDATGO
: Die lokale TS-Anwendung darf auf der in *tref* angegebenen Verbindung wieder Vorrangdaten senden.

  Mögliche Reaktion: *x_xdatrq()*.

  NEABX zeigt dieses Ereignis nur an, wenn beim Verbindungsaufbau die Benutzung von Vorrangdaten vereinbart wurde.

  Normaldaten dürfen weiterhin nicht gesendet werden.

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 bereits aufgebaute 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. Die von der gerufenen TS-Anwendung beim *x_conrs()* übergebenen Benutzerdaten müssen noch entgegengenommen werden.

  Von NEABX erwartete Reaktion: Wiederholung des *x_concf()*-Aufrufs.

X_REPCIN
: Die Benutzerdaten, die die rufende TS-Anwendung bei der Verbindungsaufforderung mitgegeben hat, müssen entgegengenommen werden.

  Von NEABX erwartete Reaktion: Wiederholung des *x_conin()*-Aufrufs.

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: Wiederholen des *x_conrq()*-Aufrufs.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

  Der Inhalt von *tref* kann gegenüber dem Inhalt beim Aufrufzeitpunkt verändert sein, ist aber in jedem Fall nicht definiert.

Fehler

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_event - Ereignis abwarten oder abfragen (event)“ auf Seite 158 aufgelisteten Fehlerwerte auftreten. Sie können durch Aufruf von x_error() abgefragt werden.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_attach(), x_concf(), x_conin(), x_datain(), x_datago(), x_datastop(), x_disin(), x_error(), x_redin(), x_xdatin(), x_xdatgo(), x_xdatstop()

9.7.15 x_info - Informationen über NEABX-Konstante (information)

x_info liefert die maximal mögliche Länge einer TIDU für die angegebene Verbindung. Die TIDU-Länge ist abhängig vom verwendeten Transportsystem. Sie brauchen sie für die Aufrufe zum Datentransport.

```
#include <cmx.h>
#include <neabx.h>
int   x_info (int *tref,
              struct x_opti1 *x_opt);
```

-> tref

Zeiger auf die Transportreferenz. Hier geben Sie die Transportreferenz der Verbindung ein, zu der Sie die maximal mögliche Länge einer TIDU wissen wollen.

<> x_opt

Zeiger auf eine Union, 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

In dieses Feld trägt NEABX die Länge der TIDU ein. Dieser Wert gibt an, wieviele Byte beim Datentransfer über die angegebene Verbindung pro Aufruf an NEABX übergeben bzw. von NEABX empfangen werden können.

x_maxl ist unverändert jener Wert, den die Funktion *t_info()* liefert. Eventuelle NEABX-Protokolllängen sind NICHT berücksichtigt. Es gelten die Regeln, die bei *x_datarq()*, *x_datain()*, *x_xdatrq()* und *x_xdatin()* bzgl. der Verwendung der Optionen X_OPTD1, X_OPTD2 und X_OPTD3 festgelegt wurden.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_info - Informationen über CMX abfragen (information)“ auf Seite 186 aufgelisteten Fehlerwerte auftreten. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

9.7.16 x_neavi - Analyse des NEABV-Protokolls

x_neavi() analysiert das NEABV-Protokoll aus einem Datenbereich, der durch die Aufrufe *x_conin()* bzw. *x_concf()* vom Netz aus versorgt wurde.

Der bei *x_conin()* bzw. *x_concf()* gelieferte Zeiger auf die Benutzerverbindungsnachricht (*x_udatap*) kann direkt an *x_neavi()* übergeben werden.

x_neavi() interpretiert die Daten der Benutzerverbindungsnachricht und schreibt die in ihr enthaltenen Informationen in die Elemente der bereitgestellten Optionsstruktur.
Die in der Optionsstruktur zurückgelieferten Adressen sind Teiladressen des übergebenen Bereichs *x_udatap*.

```
int   x_neavi (char *x_udatap,
               int *x_udatal,
               x_optneav *x_opt);
```

-> 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()* bzw. *x_conin()* gelieferten Datenbereich *x_udatap* weiterzureichen.

*x_udatal
: Zeiger auf einen Bereich, der die Länge des Datenbereiches *x_udatap* beschreibt.

  Es ist sinnvoll, den von *x_concf()* bzw. *x_conin()* gelieferten Wert *x_udatal* weiterzureichen.

<> x_opt
: Zeiger auf eine Union *x_optneav*, in die NEABX die Struktur *x_optrk* einträgt. Die Struktur enthält die Ergebnisse der Analyse der NEABV-Protokolldaten, die in *x_udatap* übergeben wurden.

  Elemente die nicht im NEABV-Protokoll enthalten sind, werden mit NULL versorgt.

  Die Angabe von *x_opt* ist obligatorisch, da NEABX sonst die Analyseergebnisse nicht an die TS-Anwendung übergeben kann.

  Die Struktur *x_optrk* und die Union *x_optneav* sind in der Datei <*neabx.h*> definiert.

```
struct   x_optrk   {
                      /* STRUKTUR x_opt BEI
                         RECHNERKOPPLUNG  */
->  int   x_optnr;
                      /* Optionsnr. = X_OPTRK, X_OPTRK1 */
<-  int   x_init;
                      /* Initiative bei Datenübermittlung */
<-  int   x_opchl;   /* Länge der OPCH in x_opchp */
<-  char  *x_opchp;  /* Zeiger auf die OPCH */
<-  int   x_bvmsgl;
                      /* Länge d. Benutzer-Verb.-Nachricht */
<-  char  *x_bvmsgp;};
                      /* Zeiger auf Benutzer-Verb.-Nachricht */
}
```

x_optnr
: Optionsnummer. Anzugeben ist:

  X_OPTRK oder X_OPTRK1

x_init
: Initiative zur Datenübertragung bei Rechnerkopplung.

  Mögliche Werte: X_MYINIT oder X_INITRQ.

  Die Bedeutung dieser Werte ist in Abschnitt „Die NEABX-Dienstfunktionen (NEABV-Service)“ auf Seite 241 beschrieben.

x_opchl
: Längen des Operation-Characters OPCH auf den *x_opchp* zeigt.

  NEABX trägt dann die im NEABV enthaltene Länge ein.

x_opchp
: Zeiger auf den Bereich, der die Operations-Characters (OPCH) enthält, oder NULL-Zeiger.

  Die aktuelle Länge wird durch *x_opchl* beschrieben.

x_bvmsgl
: Längenangabe zu dem Bereich *x_bvmsgp*.

  NEABX trägt die im NEABV-Protokoll enthaltene Länge ein.

x_bvmsgp
: Zeiger auf einen Bereich, der die NEABV-Benutzerverbindungsnachricht enthält, oder NULL-Zeiger.

  Die aktuelle Länge wird durch *x_bvmsgl* beschrieben.

  Die Nachricht wird von *x_neavi()* nicht umcodiert oder behandelt.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Die Optionsstruktur enthält die Ergebnisse der Analyse des NEABV-Protokolls

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_NVERR2
: Der in *x_optnr* angegebene Wert ist falsch.

X_NVERR3
: Die Längenangabe für eines der Protokollelemente im NEABV-Protokoll ist zu groß.

X_NOOPT
: Es wurde kein Zeiger *x_opt* angegeben.

Siehe auch

x_conin(), x_concf(), x_neavo()

9.7.17 x_neavo - Erzeugen des NEABV-Protokolls

x_neavo() erzeugt das NEABV-Protokoll und hinterlegt es in einem Speicher. Der Speicher kann anschließend dem Aufruf *x_conrq()* bzw. *x_conrs()* zur Verfügung gestellt werden.

Die zum Aufbau des NEABV-Protokolls benötigten Parameter müssen in der Struktur *x_opt* versorgt werden.

```
int   x_neavo (char *x_udatap,
               int *x_udatal,
               x_optneav *x_opt);
```

<- x_udatap
: Zeiger auf einen Bereich, in den *x_neavo()* die erzeugten NEABV-Protokolldaten einträgt.

  Die Größe des Bereiches wird in *x_udatal* angegeben.

<> x_udatal
: Länge des Bereiches *x_udatap*.

  Vor dem Aufruf ist hier die Länge des zur Verfügung gestellten *x_udatap*-Bereiches einzutragen. Die maximal benötigte Länge beträgt 109 byte.

  Nach dem Aufruf trägt NEABX die Länge der in *x_udatap* eingetragenen Daten ein. Diese Länge setzt sich zusammen aus der Länge des NEABV-Protokolls und eventuell aus der Reserve für das NEABX-Protokoll. Ob die Reserve für das NEABX-Protokoll berücksichtigt wird, ist abhängig von der in *x_optnr* angegebenen Optionsnummer.

  Das Ergebnis in *x_udatap* und *x_udatal* kann direkt bei den Aufrufen *x_conrq()* bzw. *x_conrs()* eingesetzt werden, wenn die bei *x_optnr* beschriebene Zuordnung (Tabelle) beachtet wird.

-> x_opt
: Zeiger auf die Union *x_optneav*. Diese enthält die Struktur *x_optrk*, die in der Datei <neabx.h> definiert ist.

  Die Angabe von *x_opt* ist obligatorisch.

```
struct    x_optrk   {
                            /* STRUKTUR x_opt BEI
                               RECHNERKOPPLUNG */
   ->  int    x_optnr;
                            /* Optionsnr. = X_OPTRK, X_OPTRK1 */
   ->  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 */
}
```

x_optnr

Optionsnummer. Anzugeben ist:

X_OPTRK oder X_OPTRK1

Wird die Option X_OPTRK1 verwendet, dann ist für die Angabe der Länge in *x_udatal* die Reserve nicht mehr zu berücksichtigen.

Konvention für die Verwendung der Optionen:

Optionsnummer bei x_neavo()	**Optionsnummer bei x_con[rq l rs]**	**Reserve für NEABX-Protokoll in x_udatal berücksichtigt**
X_OPTRK	X_OPTC1	JA
X_OPTRK1	X_OPTC3	NEIN

x_init

Initiative zur Datenübertragung bei Rechnerkopplung.

Mögliche Angaben: X_MYINIT oder X_INITRQ.

Wenn andere als die vorgesehenen Werte angegeben werden, wird ein Fehler gemeldet (siehe auch Abschnitt „Die NEABX-Dienstfunktionen (NEABV-Service)“ auf Seite 241).

x_bvmsgp

Zeiger auf einen Bereich, der die NEABV-Benutzerverbindungsnachricht enthält.

Die gefüllte Länge wird durch *x_bvmsgl* beschrieben.

x_bvmsgl
: Längenangabe zu dem Bereich *x_bvmsgp*.

  Die maximal erlaubte Länge ist X_BVMMXL.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich. Das NEABV-Protokoll wurde in *x_udatap* hinterlegt. *x_udatal* enthält seine Länge.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_NVERR1
: Der bereitgestellte Speicherbereich *x_udatap* reicht nicht aus, um das *x_opt* entsprechende NEABV-Protokoll zu hinterlegen. Die Länge *x_udatal* ist zu klein.

X_NVERR2
: Für *x_optnr* wurde ein falscher Wert angegeben.

X_NVERR4
: Bei Rechnerkopplung: Für *x_init* wurde ein falscher Wert angegeben.

X_NVERR5
: Unzulässige Angabe zu *x_opchp, x_opchl, x_bvmsgp, x_bvmsgl, x_npwp* oder *x_npwl*.

Siehe auch

x_conrq(), x_conrs(), x_neavi()

9.7.18 x_perror - NEABX-Fehlermeldung decodiert ausgeben

x_perror() decodiert NEABX-Fehlermeldungen, die dem Prozess beim Aufruf von *x_error()* in sedezimaler Darstellung von NEABX übergeben wurden.

x_perror() schreibt den Klartext zu der in *errcod* angegebenen NEABX-Fehlermeldung auf die Standardfehlerausgabe *stderr*.

Die Funktion kann nur dann den Klartext zu einem Fehler ausgeben, wenn der Fehlertyp X_BX3 und die Fehlerklasse X_NEAERR ist.

Der gelieferte Klartext setzt sich zusammen aus dem Fehlersymbol, wie in *<neabx.h>* definiert und einem erklärenden Text. Die erklärenden Texte werden aus einem Messagekatalog des Native Language Supports (NLS) entnommen, sofern vorhanden.

Im Parameter *s* kann ein zusätzlicher erklärender Text angegeben werden, z.B Angabe auf welchen NEABX-Aufruf und welche TS-Anwendung sich der Fehler bezieht. Der Text muss als String übergeben werden („s“).

Aufbau der Ausgabe von *x_perror()*:

x_perror() schreibt zunächst den durch *s* angegebenen Text (sofern s != NULL), dann : (Doppelpunkt) und \n (Zeilenende).

Die Klartextdarstellung der Fehlermeldung erfolgt dann in drei Zeilen:

```
\t<FEHLERTYP-Symbol> <text aus msgcat>\n
\t<FEHLERKLASSE-Symbol> <text aus msgcat>\n
\t<FEHLERWERT-Symbol> <text aus msgcat>\n
```

(msgcat = Messagekatalog.)

Für nicht decodierbare Fehlerwerte wird ein „?“ ausgegeben.

```
#include <cmx.h>
#include <neabx.h>
void   x_perror (char *s,
                 int errcod);
```

-> s

Zeiger auf einen Bereich mit dem Text, der der Klartextdarstellung des Fehlers vorangestellt werden soll.

-> errcod

Für *errcod* ist die Darstellung der Fehlermeldung anzugeben, die dem Prozess beim Aufruf *x_error()* von NEABX übergeben wurde.

Beispiel

1. *x_conin()* kehrt mit Rückgabewert X_ERROR zurück.

   Der Aufruf von *x_error()* liefert den sedezimalen Wert 0x9b0e.

   Der Aufruf

   ```
   x_perror ( "x_conin" , 0x9b0e)
   ```

   decodiert diesen Fehler wie folgt:

   ```
   x_conin:
     X_BX3 Fehler von ICMX(NEA)
     X_NEAERR Fehlermeldung des Migrationsservices NEABX
     X_BADPRPI x_conin,x_concf: Protokoll-ID (=PI) Byte
     falsch
   ```

2. *x_datarq()* kehrt mit Rückgabewert X_ERROR zurück.

   Der Aufruf von *x_error()* liefert den sedezimalen Wert 0x9b3b.

   Der Aufruf

   ```
   x_perror ( "x_datarq" , 0x9b3b)
   ```

   decodiert diesen Fehler wie folgt:

   ```
   x_datarq:
       X_BX3 Fehler von ICMX(NEA)
       X_NEAERR Fehlermeldung des Migrationsservices NEABX
       X_WXOPT x_opt Angabe falsch in Bezug auf
       X_MORE/X_END, mit/ohne NEABX
   ```

9.7.19 x_redin - 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.

Der Aufruf *x_redin()* liefert:

– die Prozess-Identifikation des rufenden Prozesses,

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

Ist der laufende Prozess in mehreren TS-Anwendungen angemeldet, so muss er selbst durch geeignete Mittel feststellen, zu welcher TS-Anwendung die umgelenkte Verbindung gehört. Ein geeignetes Mittel sind die Benutzerdaten.

```
#include <cmx.h>
#include <neabx.h>
int   x_redin (int *tref,
               int *pid,
               struct x_optc2 *x_opt);
```

-> tref
: Zeiger auf die Transportreferenz. Hier tragen Sie die Transportrefenz ein, die Sie beim Aufruf *x_event()* mit Ergebnis X_REDIN erhalten haben.

<- pid
: Zeiger auf die Prozess-Nummer. NEABX trägt hier die Nummer des Prozesses ein, der die Verbindung umlenkt.

<> x_opt
: Zeiger auf die Struktur *x_optc2*. Mit dieser Struktur können Sie die Information abfragen, die der umlenkende Prozess beim Aufruf *x_redrq()* mitgeliefert hat.

  Die Angabe für *x_opt* ist obligatorisch, da der Migrationsservice immer in der Lage sein muss, eine Nachricht (das interne x_red-Protokoll) zu empfangen.

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 Datenpuffers */
    } ;
```

x_optnr
: Optionsnummer. Anzugeben ist X_OPTC2.

x_udatap
: Zeiger auf einen Bereich, in den NEABX die Benutzerdaten einträgt.

x_udatal
: Vor dem Aufruf geben Sie die Länge des bereitgestellten Datenbereichs *x_udatap* an. Der Bereich muss die Benutzerdaten aufnehmen können und noch eine Reserve von X_RED_PL Zeichen für das x_red-Protokoll enthalten. Für die Länge ist X_RED_SIZE anzugeben.

  Nach dem Aufruf enthält *x_udatal* die Nettolänge der Benutzerdaten.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

X_BADREDIR
: Die in *x_udatal* angegebene Länge ist zu klein.

X_NOINFO
: Keine TIDU-Länge bestimmbar. Die Verbindung wurde wieder abgebaut.

X_NOOPT
: Es wurde kein *x_opt*-Zeiger angegeben.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_redin - Umgelenkte Verbindung annehmen (redirection indication)“ auf Seite 190 aufgelisteten Fehlerwerte auftreten.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_error(), x_event(), x_disrq(), x_redrq()

9.7.20 x_redrq - 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 dem annehmenden Prozess Benutzerdaten mitgeschickt werden. In den Benutzerdaten kann der laufende Prozess dem annehmenden Prozess mitteilen, zu welcher TS-Anwendung die Verbindung gehört.

```
#include <cmx.h>
#include <neabx.h>
int   x_redrq (int *tref,
               int *pid,
               struct t_optc2 *x_opt);
```

-> tref

Zeiger auf die Transportreferenz. Sie tragen hier die Transportreferenz der Verbindung ein, die Sie umlenken wollen.

-> pid

Zeiger auf die Prozess-Nummer des gerufenen Prozesses. Hier geben Sie die Nummer des Prozesses an, zu dem Sie die Verbindung umlenken wollen.

-> x_opt

Zeiger auf die Struktur *x_optc2*. Mit dieser Struktur können Sie beim Verbindungsumlenken dem gerufenen Prozess Benutzerdaten mitschicken, die er beim Aufruf *x_redin()* erhält.

Die Angabe von *x_opt* ist obligatorisch, da der Migrationsservice immer eine Nachricht (internes x_red-Protokoll) übergeben muss.

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 Datenpuffers */
    };
```

x_optnr

Optionsnummer. Anzugeben ist X_OPTC2.

x_udatap

Zeiger auf einen Bereich mit Benutzerdaten, die NEABX dem annehmenden Prozess übergeben soll, plus Platzreservierung für das interne x_red-Protokoll. Das Protokoll hat eine Länge von X_RED_PL Zeichen (z.Zt. 5) und 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 – X_RED_PL.

Minimale Angabe: X_RED_PL.

Rückgabewert

T_OK

Der Aufruf war erfolgreich.

X_IMPOSSIBLE

Die Verbindung kann zur Zeit nicht umgelenkt werden, da entweder die Verbindung noch nicht vollständig aufgebaut ist, oder noch Transportquittungen an die Partner-TS-Anwendung gesendet werden müssen.

X_ERROR

Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_udatal*

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

X_NOOPT
: Es wurde kein *x_opt*-Zeiger angegeben.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_redrq - Verbindung umlenken (redirection request)“ auf Seite 194 aufgelisteten Fehlerwerte auftreten.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_datain(), x_error(), x_event(), x_xdatin()

9.7.21 x_setopt - Optionen in CMX_NEA schalten (set options)

Mit *x_setopt* können Optionen an- und ausgeschaltet werden.

Die Funktion kann zum An- und Ausschalten des ICMX(NEA) Bibliotheksverfolgers genutzt werden.

```
#include <cmx.h>
#include <neabx.h>
int x_setopt (int level,
              x_opts *opt);
```

-> level

Gibt an, in welcher ICMX(NEA)-Komponente die Option gesetzt werden soll.

Mögliche Werte:

X_LIB

Bei der gesetzten Option handelt es sich um eine ICMX(NEA) Bibliotheksoption.

-> opt

Zeiger auf eine Union, die eine Optionsstruktur enthält.

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

```
        typedef union x_optset {
        struct x_opts1 opts1 ;   /* Steuerstruktur */
        } x_opts ;

        struct x_opts1 {
->      int x_optnr;             /* Options-Nr.  */
->      int x_optname;           /* Options-Name */
->      char *x_optvalue;        /* Zeiger auf Optionen String */
    };
```

t_optnr

Optionsnummer. Anzugeben ist X_OPTS1.

t_optname

Gibt die Option an, die an- oder ausgeschaltet werden soll.

Mögliche Werte:

X_DEBUG

Der Bibliotheksverfolger soll dem Inhalt des Elements *x_optvalue* entsprechend geschaltet werden.

x_optvalue
: Zeiger zu einem String (abgeschlossen mit NIL), der Angaben zu Art und Umfang des zu aktivierenden Schnittstellenverfolgers beinhaltet. Das Format ist identisch zu dem der Umgebungsvariablen NEATRACE im Kommando *neal* (siehe Handbücher „CMX, Betrieb und Administration“ [1] und [2]).

Rückgabewert

X_OK
: Der Aufruf war erfolgreich.

X_ERROR
: Die Option wurde nicht gesetzt.

Fehler

Mit der Funktion *x_error* kann der Fehlerwert ermittelt werden. Dieser hat bei Nichteinhaltung der Wertebereiche der einzelnen Parameter den Typ X_B3, die Klasse X_NEAERR und den Wert X_WPARAMETER. Andere Fehlerwerte sind Systemaufruffehler und entsprechen den Werten von *errno*, die in *<errno.h>* definiert sind.

9.7.22 x_strerror - NEABX-Fehlermeldung decodieren

x_strerror() decodiert NEABX-Fehlermeldungen, die dem Prozess beim Aufruf von *x_error()* in sedezimaler Darstellung von NEABX übergeben wurden. *x_strerror()* decodiert Fehlermeldungen mit Fehlertyp X_XB3 und Fehlerklasse X_NEAERR.

x_strerror() liefert den Zeiger auf einen statischen Bereich, der die Klartextdarstellung zu der in *errcod* angegebenen NEABX-Fehlermeldung enthält. Es ist zu beachten, dass *x_strerror()* bei jedem Aufruf den gleichen Speicherbereich für die Klartextübergabe verwendet.

Der Klartext setzt sich zusammen aus Fehlersymbolen, wie in <*neabx.h*> definiert, und begleitendem Text. Jedes Fehlersymbol wird von \t eingeleitet. Jeder begleitende Text wird mit \n abgeschlossen.

Die erklärenden Texte werden aus einem Messagekatalog des Native Language Supports (NLS) entnommen, sofern vorhanden.

```
#include <cmx.h>
#include <neabx.h>
char   *x_strerror (int errcod);
```

-> errcod

Für *errcod* ist die Darstellung der Fehlermeldung anzugeben, die dem Prozess beim Aufruf von *x_error()* von NEABX übergeben wurde.

Rückgabewert

War der Aufruf erfolgreich, so liefert *x_strerror()* den Zeiger auf einen Bereich mit der Klartextdarstellung der NEABX-Fehlermeldung als String in C-Notation zurück.

Wird in *errcod* ein nicht definierter Wert angegeben, so liefert *x_strerror()* den Klartext:

```
"\t<errcod> Nicht decodierbar\n"
```

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

Siehe auch

x_error(), x_perror()

9.7.23 x_xdatgo - 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.

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. Normaldaten sind von *x_xdatgo()* nicht betroffen.

Der Aufruf von *x_xdatgo()* ist nur erlaubt, wenn beim Verbindungsaufbau der Austausch von Vorrangdaten vereinbart wurde.

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

-> tref

Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, für die Sie den Vorrangdatenfluss freigeben wollen.

Rückgabewert

T_OK

Der Aufruf war erfolgreich.

X_ERROR

Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR kann der folgende Fehlerwert auftreten:

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_xdatgo - Vorrangdatenanzeige freigeben (expedited data go)“ auf Seite 209 aufgelisteten Fehlerwerte und der folgende Fehlerwert auftreten:

T_WSEQUENCE
: Die in *tref* angegebene Verbindung ist nicht vollständig aufgebaut.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_event(), x_error(), x_xdatstop()

9.7.24 x_xdatin - 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.

Wenn Sie nicht bereit sind, Vorrangdaten zu empfangen, können Sie mit *x_xdatstop()* den Vorrangdatenfluss stoppen. Damit verhindern Sie, dass NEABX dem laufenden Prozess das Ereignis X_XDATIN zustellt. Einmal mit X_XDATIN angezeigte Vorrangdaten müssen Sie jedoch immer vollständig abholen.

```
#include <cmx.h>
#include <neabx.h>
int   x_xdatin (int *tref,
                char *x_datap,
                int *x_datal,
                x_optd *x_opt);
```

-> 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.

<- 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 der angegebenen Optionsnummer entsprechend behandelt:

X_OPTD1:

Der Speicherbereich muss eine Reserve für das NEABX-Protokoll enthalten (kompatibler Modus zu CMX V2.1).

X_OPTD2:

Das NEABX-Protokoll wird für die TS-Anwendung nicht sichtbar.

X_OPTD3:

Das NEABX-Protokoll wird an den Beginn des Datenbereiches gestellt und im Element *x_offset* der Option *x_optd3* wird die Länge des Protokolles hinterlegt, und somit der TS-Anwendung mitgeteilt, wo die Nettonachricht beginnt.

<- 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.

<> x_opt

Zeiger auf eine Union *x_optd*, die eine der Strukturen *x_optd1* oder *x_optd3* oder die Angabe NULL enthält.

Die Angabe von *x_opt* ist obligatorisch, wenn beim Verbindungsaufbau vereinbart wurde, dass in der Datenphase mit NEABX-Protokoll gearbeitet wird.

Die Angabe NULL ist obligatorisch, wenn beim Verbindungsaufbau vereinbart wurde, dass in der Datenphase ohne NEABX-Protokoll gearbeitet wird.

Die Strukturen *x_optd1* und *x_optd3* und die Union *x_optd* sind in der Datei *<neabx.h>* definiert.

```
    struct x_optd1 {
->      int   x_optnr;
                              /* Optionsnummer,X_OPTD1,X_OPTD2 */
<-      int   x_code;         /* Nachrichtencode */
<-      int   x_strukt;       /* Nachrichtenstruktur */
<-      int   x_quit;         /* Transportquittungen */
<-      short x_seqno;        /* Nachrichtensequenznummer */

    struct x_optd3 {
->      int   x_optnr;        /* Optionsnummer,X_OPTD3 */
<-      int   x_code;         /* Nachrichtencode */
<-      int   x_strukt;       /* Nachrichtenstruktur */
<-      int   x_quit;         /* Transportquittungen */
<-      short x_seqno;        /* Nachrichtensequenznummer */
<-      int   x_offset;
                              /* Offset bis zum Beginn der Daten */
    };
```

x_optnr
: Optionsnummer. Mögliche Angaben:

  X_OPTD1 oder X_OPTD2 bei *x_optd1*

  X_OPTD3 bei *x_optd3*

  Die Bedeutung der Werte ist bei *x_datap* beschrieben.

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.

x_quit
: ist nur relevant, wenn beim Verbindungsaufbau die Behandlung von Transportquittungen in der TS-Anwendung angegeben wurde.

  Wurde ein DATA-Protokollelement empfangen, sind folgende Werte für *x_quit* möglich:

  0 keine Quittung verlangt.

  1 Transportquittung wird verlangt.

x_seqno
: enthält die Nachrichtensequenznummer, sofern *x_quit* nicht NULL ist.

x_offset
: In dieses Feld liefert NEABX die Länge des NEABX-Protokolls zurück. Der TS-Anwendung wird so mitgeteilt, wo die Nettonachricht beginnt.

Rückgabewert

T_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.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_datal*

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

X_NOTDTPE
: DATA-Protokollelement erwartet, aber nicht erhalten.

X_QUITPE
: *x_xdatin()* hat ein QUITTUNG-Protokollelement erhalten. Transportquittungen werden aber nicht als Vorrangdaten erwartet.

X_BADDTPELI
: Die im empfangenen NEABX-Protokoll angegebene Länge des DATA-Protokollelements stimmt nicht.

X_BADXREAD
: *x_xdatin()* liefert Wert größer 0 zurück. Der angegebene Datenbereich reichte nicht aus um die gesamte Nachricht zu übernehmen. Es stehen noch Daten zu dieser Nachricht an.

X_NOTDTPE
: DATA-Protokollelement erwartet, aber nicht erhalten.

X_WPARAMETER
: Fehlerhafter Parameter, es wurde ein falscher Wert für *x_optnr* angegeben.

X_WXOPT
: Falsche *x_opt* Angabe:

  x_opt != NULL, obwohl kein NEABX-Protokoll;

  x_opt = NULL, obwohl NEABX-Protokoll vereinbart.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_xdatin - Vorrangdaten empfangen (expedited data indication)“ auf Seite 211 aufgelisteten Fehlerwerte auftreten.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_error(), x_event()

9.7.25 x_xdatrq - 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.

Vorrangdaten unterliegen einer eigenen Fluss-Regelung, 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.

```
#include <cmx.h>
#include <neabx.h>
int   x_xdatrq (int *tref,
                char *x_datap,
                int *x_datal,
                x_optd *x_opt);
```

-> tref

Zeiger auf die Transportreferenz. Hier geben Sie die Transportreferenz der Verbindung an, auf der Sie Vorrangdaten senden wollen.

-> 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.

-> x_datal

Zeiger auf eine Längenangabe **x_datal*. **x_datal* darf niemals größer als X_EXP_SIZE sein und hat folgende Bedeutung:

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* (1 byte Sendedaten): **x_datal* = 1.

Fall *x_opt* != NULL (siehe *x_opt*) mit Optionsnummer:

X_OPTD1:

**x_datal* muss um X_DRQPHL größer angegeben werden, als es den zu sendenden Nettodaten entspricht. Der Speicherbereich der TS-Anwendung wird jedoch nicht für den Aufbau des Protokolles herangezogen.

Maximale Angabe in **x_datal* = X_EXP_SIZE.

Maximum von Benutzerdaten pro *x_xdatrq()*:
X_EXP_SIZE - X_DRQPHL

Minimale Angabe in **x_datal* (1 byte Sendedaten) =
**x_datal* = 1 + X_DRQPHL.

X_OPTD2:

**x_datal* enthält nur die Nettodatenlänge. Der Speicherbereich der TS-Anwendung muss keinerlei Reserven für das NEABX-Protokoll berücksichtigen.

Maximale Angabe in **x_datal* = X_EXP_SIZE – X_DRQPHL.

Minimale Angabe in **x_datal* = 1 (1 byte Sendedaten).

X_OPTD3:

**x_datal* enthält nur die Nettodatenlänge. Im Element *x_offset* der Optionsstruktur hinterlegt die TS-Anwendung die Distanz von *x_datap* weggerechnet, ab der die Nettodaten beginnen. Diese Distanz muss X_DRQPHL sein.

Maximale Angabe in **x_datal* = X_EXP_SIZE – X_DRQPHL.

Minimale Angabe in **x_datal* (1 byte Sendedaten) = **x_datal* = 1 + X_DRQPHL.

-> x_opt

Zeiger auf eine Union, die eine der Strukturen *x_optd1*, *x_optd3* oder die Angabe NULL enthält. Die Angabe NULL ist obligatorisch und nur dann erlaubt, wenn beim Verbindungsaufbau vereinbart wurde, dass in der Datenphase ohne NEABX-Protokoll gearbeitet wird.

Die Strukturen *x_optd1* und *x_optd3* und die Union *x_optd* sind in der Datei *<neabx.h>* definiert.

```
    struct x_optd1 {
->      int   x_optnr       /* Optionsnummer,X_OPTD1,X_OPTD2 */
->      int   x_code;       /* Nachrichtencode */
->      int   x_strukt;     /* Nachrichtenstruktur */
->      int   x_quit;       /* Transportquittungen */
->      short x_seqno;      /* Nachrichtensequenznummer */
    };

    struct x_optd3 {
->      int   x_optnr;      /* Optionsnummer,X_OPTD3 */
->      int   x_code;       /* Nachrichtencode */
->      int   x_strukt;     /* Nachrichtenstruktur */
->      int   x_quit;       /* Transportquittungen */
->      short x_seqno;      /* Nachrichtensequenznummer */
->      int   x_offse       /* Offset bis zum Beginn der Daten */
    };
```

x_optnr

Optionsnummer. Mögliche Angaben:

X_OPTD1 oder X_OPTD2 bei *x_optd1*

X_OPTD3 bei *x_optd3*

Die Bedeutung der Werte ist bei *x_datal* beschrieben.

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.

x_quit
: ist nur relevant, wenn beim Verbindungsaufbau die Behandlung von Transportquittungen in der TS-Anwendung angegeben wurde.

  Das Quittungsanforderungsbit QVBIT wird in das NEABX-Protokoll gesetzt.

x_seqno
: enthält die Nachrichtensequenznummer, sofern *x_quit* nicht Null ist.

x_offset
: Im Element *x_offset* hinterlegt die TS-Anwendung die Distanz von *x_datap* weggerechnet, ab der die Nettodaten beginnen. Diese Distanz muss X_DRQPHL sein.

Rückgabewert

T_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.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR können folgende Fehlerwerte auftreten:

X_BADLEN
: Ungültige Datenpufferlänge in *x_datal*

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

X_BADXCODE
: Der Wert in *x_code* ist falsch.

X_WPARAMETER
: Fehlerhafter Parameter, für *x_optnr* wurde ein falscher Wert angegeben.

X_WXOPT
: Falsche *x_opt* Angabe:

  x_opt != NULL, obwohl kein NEABX-Protokoll;

  x_opt = NULL, obwohl NEABX-Protokoll vereinbart.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_xdatrq - Vorrangdaten senden (expedited data request)“ auf Seite 213 aufgelisteten Fehlerwerte und der folgende Fehlerwert auftreten.

T_WSEQUENCE
: Die in *tref* angegebene Verbindung ist noch nicht vollständig aufgebaut.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_error(), x_event(), x_xdatstop()

9.7.26 x_xdatstop - 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.

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. Sie können auch auf der angegebenen Verbindung Daten senden. *x_xdatatop()* setzt nur eine Datenempfangssperre.

– 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.

Wenn Sie bei *x_event()* eines der Ereignisse X_XDATIN oder X_DATAIN erhalten haben, müssen Sie die anstehenden Daten zunächst vollständig lesen. *x_xdatstop()* bewirkt nur, dass kein weiteres Ereignis X_XDATIN oder X_DATAIN zugestellt wird.

Erneut freigeben können Sie den Vorrangdatenfluss mit *x_xdatgo*.

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

-> tref
: Zeiger auf die Transportreferenz. Hier tragen Sie die Transportreferenz der Verbindung ein, für die Sie den Vorrangdatenfluss stoppen wollen.

Rückgabewert

T_OK
: Der Aufruf war erfolgreich.

X_ERROR
: Fehler. Fehlercode mit *x_error()* abfragen.

Fehler

Im Fehlerfall sind die folgenden Fehlerwerte möglich. Sie können durch Aufruf von *x_error()* abgefragt werden.

Zu Fehlertyp X_BX3 und Fehlerklasse X_NEAERR kann der folgende Fehlerwert auftreten:

X_BADTABLE
: Die angegebene Transportreferenz *tref* ist dem Migrationsservice nicht bekannt. Sie ist in der betreffenden Tabelle nicht vorhanden.

Zu Fehlertyp T_CMXTYPE und Fehlerklasse T_CMXCLASS können die in Abschnitt „t_xdatstop - Vorrangdatenanzeige sperren (expedited data stop)“ auf Seite 216 aufgelisteten Fehlerwerte und der folgende Fehlerwert auftreten.

T_WSEQUENCE
: Die in *tref* angegebene Verbindung ist noch nicht vollständig aufgebaut.

Zusätzlich können die bei *ioctl(2)* aufgelisteten Fehler auftreten.

Siehe auch

x_datago(), x_error(), x_event(), x_xdatgo(), x_xdatrq()

10 Anhang

10.1 Liste aller CMX-Fehlermeldungen

Die folgenden Tabellen enthalten alle möglichen CMX-Fehlermeldungen, d. h. alle Fehlermeldungen, die an den Programmschnittstellen ICMX(L) und ICMX(NEA) erzeugt werden. Die Fehlermeldungen sind nach Fehlertyp und Fehlerklasse sortiert.

Fehlermeldungen mit Fehlerklasse = T_DSSYSERR (5) bzw. X_BX2 (8) sind nicht in den Include-Dateien der CMX-Programmschnittstellen definiert. Sie können nur mit <*errno.h*> entschlüsselt werden.

An ICMX(L) erzeugte Fehlermeldungen

Fehlerwerte mit Fehlertyp (0) = T_CMXTYPE und
Fehlerklasse = T_CMXCLASS (0):

Num. Wert	Symbolischer Wert	Bedeutung
0	T_NOERROR	Kein Fehler
4	T_ENOENT	Alle intern bereitgestellten Resourcen sind belegt.
5	T_EIO	Das CCP ist nicht mehr betriebsbereit.
12	T_ENOMEM	Der Arbeitsspeicher reicht nicht aus.
14	T_EFAULT	Unzulässige Adresse. Einer der angegebenen Zeiger zeigt nicht in den Prozess-Adressraum.
22	T_EINVAL	Unzulässiges Argument
100	T_UNSPECIFIED	Nicht näher spezifizierter Fehler
101	T_WSEQUENCE	Der Funktionsaufruf ist in diesem Zustand unzulässig.
102	T_WREQUEST	Unzulässiger Funktionsaufruf Bei Kopplung an SNA: Im Prozess ist nur der Platzhaltermodul eingebunden.
103	T_WPARAMETER	Fehlerhafter Parameter

Tabelle 10: An ICMX erzeugte Fehlermeldungen

Num. Wert	Symbolischer Wert	Bedeutung
104	T_WAPPLICATION	Die Anwendung ist unbekannt oder die Anwendung ist bereits unter diesem Namen bekannt.
105	T_WAPP_LIMIT	Es ist keine Anmeldung von Prozessen in Anwendungen mehr möglich.
106	T_WCONN_LIMIT	Grenzwert für Verbindungen erreicht
107	T_WTREF	Unzulässige Transportreferenz
109	T_COLLISION	Kollision bei Verbindungsaufbau, -abbau, -umlenkung bzw. beim Senden von Daten
110	T_WPROC_LIMIT	Zu viele Prozesse haben Anwendungen angemeldet.
111	T_NOCCP	Kein CCP für gewünschte Anmeldung oder Verbindung vorhanden
112	T_ETIMEOUT	Wartezeit abgelaufen
113	T_WROUTINFO	Unzulässige CC-Liste
115	T_WRED_LIMIT	Zu viele simultane Verbindungsumlenkungen
116	T_WLIBVERSION	Version der verwendeten CMX-Bibliothek inkompatibel
117	T_CBRECURSIVE	Rekursiver Aufruf von *t_event* nicht erlaubt
118	T_W_NDS_ACCESS	Ein NDS-Fehler ist aufgetreten
119	T_EMUTEX	Ein Fehler ist beim Mutex-Handling aufgetreten
120	T_NOTSD	Fehler bei der Zuweisung thread-spezifischen-Speichers

Tabelle 10: An ICMX erzeugte Fehlermeldungen

An ICMX(L) können zusätzlich die folgenden Fehlerwerte auftreten, die nicht zu Fehlertyp = T_CMXTYPE und Fehlerklasse = T_CMXCLASS gehören.

Num. Wert	Symbolischer Wert	Bedeutung
0	T_NOTSPEC	Nicht näher spezifizierter Fehler
1	T_DIRERR	Das angegebene TS-Directory ist unbekannt.
2	T_NAMERR	Der angegebene Name ist nicht vorhanden oder der angegebene Name ist bereits vorhanden.
3	T_ILLNAM	Der Name ist syntaktisch falsch.
5	T_PROPER	Die abgefragte Eigenschaft ist nicht vorhanden oder die angegebene Eigenschaft ist bereits vorhanden oder syntaktisch falsch.
7	T_TIMOUT	Der TNSX-Dämon *tnsxd* antwortet nicht im Zeitlimit.
10	T_LEAFNO	Es wurden mehr bzw. weniger Namen als erwartet gefunden.
16	T_LENERR	Eigenschaften hat eine falsche Länge.
20	T_PROT	Fehler im Protokoll zu *tnsxd*
100	T_LFILE	Das TS-Directory hat ein falsches Format.

Tabelle 11: Zusätzliche Fehlerwerte an ICMX(L)

An ICMX(NEA) erzeugte Fehlermeldungen

Fehlerwerte mit Fehlertyp = X_BX3 (9) und Fehlerklasse = X_NEAERR (0xb)

num. Wert	symbolischer Wert	Bedeutung
0	X_NOERROR	Kein Fehler
1	X_DIDSSSTK	Partner hat weder ETB noch ETX bei DSS-Anschluss angegeben
3	X_MAXDAT	Beim Verbindungsaufbau mehr als X_MSG_SIZE Benutzerdaten erhalten
4	X_BADLEN	Ungültige Datenpufferlänge

Tabelle 12: An ICMX(NEA) erzeugte Fehlermeldungen

num. Wert	symbolischer Wert	Bedeutung
5	X_BADTRANS	Ungültiges Transportsystem
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
20	X_BADPVBYTE	Fehlerhaftes Protokoll-Versions-Byte erhalten
21	X_NONEBYTE	Bei Stationskopplung 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*
25	X_BADREDIR	Ungültige Verbindungsumlenkung

Tabelle 12: An ICMX(NEA) erzeugte Fehlermeldungen

num. Wert	symbolischer Wert	Bedeutung
42	X_BADMSGLEN	Ungültige Benutzernachrichtenlänge
43	X_NOXDRDSS	DSS darf *x_xdatrq()* nicht aufrufen
44	X_NOXDIDSS	DSS darf *x_xdatin()* nicht aufrufen
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* Zeiger angegeben
50	X_WPARAMETER	Fehlerhafter Parameter, z. B. falsche *x_optnr* angegeben
51	X_NVERR1	Die in *x_udatal* bei *x_neavo()* angegebene Länge ist zu kurz. NEABV-Protokoll kann nicht hinterlegt werden.
52	X_NVERR2	Unzulässige Optionsnummer in *x_neavi()*, *x_neavo()*
53	X_NVERR2	Unzulässige Längenangabe im NEABV-Protokoll in *x_neavi()*
54	X_NVERR4	Unzulässige Angabe zu *x_init* in *x_neavo()*
55	X_NVERR5	Unzulässige Angabe zu *x_opch, x_bvmsg, x_npw in x_neavo()*
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();*

Tabelle 12: An ICMX(NEA) erzeugte Fehlermeldungen

10.2 Liste der Verbindungsabbaugründe

Im Folgenden sind die von CMX bei den Aufrufen *t_disin()* und *x_disin()* in *reason* übergebenen Verbindungsabbaugründe beschrieben. Die hier angegebenen symbolischen Werte sind in <*cmx.h*> numerisch definiert. Die Abkürzung CCP steht hier für "Communication Control Program" und gemeint ist damit das Transportsystem. „Lokales CCP" steht für das CCP im System des laufenden Prozesses, „Partner-CCP" für das CCP im System des Verbindungspartners des laufenden Prozesses.

Von CMX angegebene Gründe

Num. Wert	Symbolischer Wert	Bedeutung
0	T_USER	Der Abbau erfolgte durch den Kommunikationspartner; u.U. auch durch einen Benutzerfehler des Kommunikationspartners.
1	T_RTIMEOUT	Wegen Inaktivität der Verbindung gemäß Parameter *t_timeout*, wurde die Verbindung lokal durch CMX abgebaut.
2	T_RADMIN	Wegen Außerbetriebnahme des CCP durch die Administration, wurde die Verbindung lokal durch CMX abgebaut.
3	T_RCCPEND	Wegen CCP-Ausfall, wurde die Verbindung lokal durch CMX abgebaut.

Tabelle 13: Verbindungsabbaugründe - von CMX angegeben

Vom Partner-CCP angegebene Gründe

Num. Wert	Symbolischer Wert	Bedeutung
256	T_RUNKNOWN	Der Partner oder das CCP hat die Verbindung abgebaut. Ein Grund für den Abbau wurde nicht angegeben.
257	T_RSAPCONGEST	Wegen eines TSAP-spezifischen Engpasses hat das Partner-CCP die Verbindung abgebaut.

Tabelle 14: Verbindungsabbaugründe - vom Partner-CCP angegeben

Num. Wert	Symbolischer Wert	Bedeutung
258	T_RSAPNOTATT	Das Partner-CCP hat die Verbindung abgebaut, weil der adressierte TSAP dort nicht angemeldet ist.
259	T_RUNSAP	Das Partner-CCP hat die Verbindung abgebaut, weil der adressierte TSAP dort nicht bekannt ist.
261	T_RPERMLOST	Abbau durch Netzadministration oder Administration des Partner-CCP.
262	T_RSYSERR	Fehler im Netz.
385	T_RCONGEST	Wegen Betriebsmittelengpass hat das Partner-CCP die Verbindung abgebaut.
386	T_RCONNFAIL	Wegen Misslingen des Verbindungsaufbaus, hat das Partner-CCP die Verbindung abgebaut. Der Verbindungsaufbau kann misslingen, weil z. B. Benutzerdaten zu lang oder Vorrangdaten nicht zugelassen sind.
387	T_RDUPREF	Weil für ein NSAP-Paar eine zweite Verbindungsreferenz vergeben wurde (Systemfehler), wurde die Verbindung vom Partner-CCP abgebaut.
388	T_RMISREF	Wegen einer nicht zuzuordnenden Verbindungsreferenz (Systemfehler), hat das Partner-CCP die Verbindung abgebaut
389	T_RPROTERR	Wegen eines Protokollfehlers (Systemfehler), hat das Partner-CCP die Verbindung abgebaut.
391	T_RREFOFLOW	Wegen Verbindungsreferenz-Überlaufs, hat das Partner-CCP die Verbindung abgebaut.
392	T_RNOCONN	Das Partner-CCP hat den Aufbau der Netzverbindung abgelehnt.
394	T_RINLNG	Wegen falscher Header- oder Parameterlänge (Systemfehler), hat das Partner-CCP die Verbindung abgebaut.

Tabelle 14: Verbindungsabbaugründe - vom Partner-CCP angegeben

Vom Partner-CCP mit aktivem Transport-Gateway (CS-GATE) angegebene Gründe

Num. Wert	Symbolischer Wert	Bedeutung
396	TG_BADADDR_TS	Transport-Gateway: Transportsystem entdeckte Konfigurierungsfehler.
397	TG_BADOPT_TS	Transport-Gateway: Transportsystem entdeckte fehlerhafte Optionen.
398	TG_SYSERR_TS	Transport-Gateway: Systemfehler in Transportsystem.
399	TG_SYSERR_FS	Transport-Gateway: Systemfehler in Forwarding-Support-Service (FSS).
400	TG_BADADDR_FS	Transport-Gateway: Konfigurierungsfehler in Forwarding-Support-Information-Base (FSB). Dieser Abbaugrund entspricht T_RSAPNOTATT in einem Partner-System, in dem das Transport-Gateway nicht aktiv ist.
401	TG_SYSERR_TGWY	Transport-Gateway: Systemfehler in Transport-Gateway-Entity (TGWY).
402	TG_BADADDR_TGWY	Transport-Gateway: Transport-Gateway-Entity (TGWY) entdeckte Konfigurierungsfehler.
403	TG_BADOPT_TGWY	Transport-Gateway: Transport-Gateway-Entity (TGWY) entdeckte fehlerhafte Optionen.

Tabelle 15: Verbindungsabbaugründe - vom Partner-CCPmit aktivem CS-Gate angegeben

Vom lokalen CCP angegebene Gründe

Num. Wert	Symbolischer Wert	Bedeutung
448	T_RLCONGEST	Wegen Betriebsmittelengpass hat das lokale CCP die Verbindung abgebaut.
449	T_RLNOQOS	Weil Quality of Service nicht mehr geboten werden kann, hat das lokale CCP die Verbindung abgebaut.
451	T_RILLPWD	Ungültiges (Verbindungs-) Passwort.
452	T_RNETACC	Netzzugang wurde verweigert.
464	T_RLPROTERR	Wegen eines Transportprotokollfehlers (Systemfehler), hat das lokale CCP die Verbindung abgebaut.
465	T_RLINTIDU	Weil es eine zu lange Schnittstellen-Dateneinheit (TIDU) erhalten hat (Systemfehler), hat das lokale CCP die Verbindung abgebaut.
466	T_RLNORMFLOW	Wegen Verletzung der Fluss-Kontrollregeln für Normaldaten (Systemfehler), hat das lokale CCP die Verbindung abgebaut.
467	T_RLEXFLOW	Wegen Verletzung der Fluss-Kontrollregeln für Vorrangdaten (Systemfehler), hat das lokale CCP die Verbindung abgebaut.
468	T_RLINSAPID	Weil es eine ungültige TSAP-Identifikation erhalten hat (Systemfehler), hat das lokale CCP die Verbindung abgebaut.
469	T_RLINCEPID	Weil es eine ungültige TCEP-Identifikation erhalten hat (Systemfehler), hat das lokale CCP die Verbindung abgebaut.
470	T_RLINPAR	Wegen eines unzulässigen Parameterwerts, z. B. Benutzerdaten zu lang oder Vorrangdaten nicht zugelassen, hat das lokale CCP die Verbindung abgebaut.
480	T_RLNOPERM	Die Administration des lokalen CCP hat den Verbindungsaufbau verhindert.

Tabelle 16: Verbindungsabbaugründe: - vom lokalen CCP angegeben

Num. Wert	Symbolischer Wert	Bedeutung
481	T_RLPERMLOST	Die Administration des lokalen CCP hat die Verbindung abgebaut.
482	T_RLNOCONN	Weil keine Netzverbindung verfügbar ist, konnte das lokale CCP den Verbindungsaufbau nicht durchführen.
483	T_RLCONNLOST	Wegen Verlust der Netzverbindung, hat das lokale CCP die Verbindung abgebaut. Häufigste Ursache: Generierungsfehler auf CCP- und PDN-Seite, z. B. unstimmige Link-Adressen. Als Problemursache kommt auch in Frage: Partner ist nicht vorhanden, Modem ist defekt oder falsch eingestellt, DÜ-Anschluss ist nicht gesteckt, DFÜ-Board ist defekt.
484	T_RLNORESP	Weil der Partner nicht auf CONRQ antwortet, ist der Verbindungsaufbau vom lokalen CCP nicht durchführbar. Häufigste Ursache: Der Solaris-Rechner wurde bei der Prozessorkopplung über WAN nicht in den Partnerrechnern bekanntgemacht. Lösung: Prozessor- und Regionsnummer des hinzugenommenen Rechners bei seinen Partnerrechnern im Netz in der KOGS eintragen.
485	T_RLIDLETRAF	Wegen Verlust der Verbindung (Idle Traffic Timeout), hat das lokale CCP die Verbindung abgebaut.
486	T_RLRESYNC	Weil die Resynchronisierung erfolglos war (mehr als 10 Wiederholungen), hat das lokale CCP die Verbindung abgebaut.
487	T_RLEXLOST	Weil der Vorrangdatenkanal defekt ist (mehr als 3 Wiederholungen), hat das lokale CCP die Verbindung abgebaut.

Tabelle 16: Verbindungsabbaugründe: - vom lokalen CCP angegeben

Fachwörter

Adresse
siehe *TRANSDATA-Adresse* und *TRANSPORTADRESSE*.

Agent
Auftragnehmer von Netzmanagement-Aufträgen.

Aktiver Partner
Der *Kommunikationspartner*, der selbst eine Verbindung zu einer anderen TS-Anwendung aufbaut.

Anwendung
Eine Anwendung ist ein System von Programmen, das ein bestimmtes Dienstangebot eines EDV-Systems anwendet, um einem menschlichen oder maschinellen Nutzer eine höherwertige Dienstleistung anzubieten. Kommunikationsanwendungen sind Anwendungen, die die Kommunikationsfunktionen eines EDV-Systems nutzen, um unter Nutzung eines Netzes systemübergreifende Dienstleistungen zu erbringen.

Den meisten Anwendungen wird ein Präfix zur Kennzeichnung des untergelagerten Dienstangebots vorangestellt (*CMX-Anwendung*, UTM-Anwendung, DCAM-Anwendung, Motif-Anwendung und Windows-Anwendung, etc.). Beispiele für Kommunikationsanwendungen sind Filetransfer, Terminalemulation, Electronic Mail, World Wide Web Browser und Server, Transaktionssysteme wie UTM, allgemein alle Anwendungen nach dem Client-/Server-Prinzip.

API (Application Programming Interface)
APIs sind Programmschnittstellen, die die Funktionen eines Programmsystems zur Verfügung stellen. Als Programmierer nutzen Sie die APIs bei der Programmierung von Anwendungen. APIs bieten Funktionen zum Verbindungsmanagement, zum Datenaustausch und zur Abbildung von Namen in Adressen. APIs im CMX-Umfeld sind ICMX, XTI und TLI.

ASCII
Internationaler Zeichensatz für DV-Systeme auf 7 Bit-Basis (ISO-7Bit-Code).

CC (Communications Controller)

Ein CC ist eine Baugruppe zum Anschluss eines Solaris-Rechners an ein Netz. Sie benötigen einen CC, um Ihren Rechner physisch an ein Subnetz anzuschließen, es sei denn, der Anschluss ist auf einer anderen Baugruppe, z. B. der Mutterplatine, mit integriert (onboard-Anschluss).

Um einen logischen Anschluss zum Netz zu erhalten, werden CCs mit dem zugehörigen Subnetzprofil geladen. Das Subnetzprofil ist Bestandteil des *CCPs*. Beispiele für ladbare CCs zum Anschluss an X.25- und Telefonnetze und ISDN sind PWXV, PWS0 und PWS2.

CCP (Communication Control Program)

Ein CCP ist ein Programmsystem, das zusammen mit einem oder mehreren *CCs* den logischen Zugang eines Solaris-Rechners an ein Netz leistet. Ein CCP implementiert die vier unteren Schichten (Transportsystem) des *OSI-Referenzmodells* zur Datenkommunikation. Ein CCP besteht aus Subnetz-Profilen und Transport Service Providern.

CMX (Communications Manager UNIX)

CMX erbringt Kommunikationsdienste zur Nutzung von *CMX-Anwendungen* und *Communication Services* im Netz und ermöglicht die Programmierung von CMX-Anwendungen. CMX vereinheitlicht die Dienste unterschiedlicher Netze und ermöglicht damit die Nutzung derselben CMX-Anwendung unabhängig vom unterliegenden Netz. Als Laufzeitsystem vermittelt CMX zwischen aktuellem Netzangebot und CMX-Anwendungen und bietet dem Netzadministrator einheitliche Funktionen für *OA&M* (Operation, Administration, Maintenance) von *CCPs* und *CCs*. Als Entwicklungssystem bietet CMX Schnittstellen (APIs) und Verfahren zur Programmierung von netzunabhängigen CMX-Anwendungen.

CMX-Anwendungen

CMX-Anwendungen sind Anwendungen, die die Dienste von CMX nutzen. CMX-Anwendungen haben im Netz eine Adresse, die *TRANSPORT-ADRESSE*. Sie identifizieren sich untereinander durch symbolische Namen, dem *GLOBALEN NAMEN* einer Anwendung.

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 TS-Anwendung im BS2000, die die Zugriffsmethode DCAM benutzt.

GLOBALER NAME einer Anwendung

Jede *CMX-Anwendung* identifiziert sich selbst und ihre Kommunikationspartner im Netz durch symbolische, hierarchische GLOBALE NAMEN. Ein GLOBALER NAME besteht aus bis zu fünf Namensteilen (NP[1- 5]), die Sie zur Definition der Anwendung (NP5), des Rechners (NP4) und (bis zu drei) administrativer Domänen (NP[3-1]) verwenden können.

Beispiel: Der GLOBALE NAME "IhreAnwendung.D018S065.mch-p.sni.de" bedeutet:

"IhreAnwendung" residiert im Host "D018S065" in der Domäne "mch-p.sni.de".

Bei der Wahl eines GLOBALEN NAMENS müssen Sie als Administrator die Vorgaben und Empfehlungen der speziellen Anwendung beachten.

Als Administrator können Sie dem GLOBALEN NAMEN einer Anwendung Eigenschaften zuordnen, zum Beispiel eine *TRANSPORTADRESSE* oder einen *LOKALEN NAMEN*. Als Programmierer können Sie die TRANSPORTADRESSE bzw. den LOKALEN NAMEN mit Hilfe der Funktionsaufrufe t_getaddr() bzw. t_getloc() aus dem GLOBALEN NAMEN gewinnen.

ICMX

Standard-Transportsystem-Schnittstelle für Anwendungen.

KOGS (Konfigurationsorientierte Generatorsprache)

KOGS ist die konfigurationsorientierte Generatorsprache, mit der die physischen und logischen Eigenschaften der Subnetz-Anschlüsse eines Rechners in einer Textdatei beschrieben werden. Sprachelemente der KOGS sind Makros, Operanden und Operandenwerte. Im Normalfall definiert der System- bzw. Netzverwalter die spezifischen Eigenschaften seiner Subnetz-Anschlüsse mit dem *CMXGUI*. Nur in Ausnahmefällen verwendet er dazu die KOGS.

Kommunikationspartner

Eine TS-Anwendung, die eine logische Verbindung zu einer anderen TS-Anwendung unterhält und Daten mit ihr austauscht.

LOKALER NAME einer Anwendung

Eine *CMX-Anwendung* meldet sich in ihrem lokalen Rechner mit dem LOKALEN NAMEN bei *CMX* zur Kommunikation an. Der LOKALE NAME besteht aus einem oder mehreren *T-Selektoren*, die jeweils das Transportsystem bezeichnen, über das die CMX-Anwendung kommunizieren soll. Als Administrator können Sie mit dem LOKALEN NAMEN die Kommunikation einer CMX-Anwendung über bestimmte Transportsysteme ermöglichen oder ausschließen und etwaige Anforderungen der CMX-Anwendung nach bestimmten T-Selektor-Werten, z. B. beim Filetransfer, erfüllen.

Den LOKALEN NAMEN einer Anwendung können Sie als Administrator dem *GLOBALEN NAMEN* der Anwendung zuordnen. Als Programmierer können Sie den LOKALEN NAMEN mit Hilfe des Funktionsaufrufs t_getloc() aus dem GLOBALEN NAMEN gewinnen.

Migrationsservice

Dienst in CMX zur Anpassung einer CMX-Anwendung an die Anforderungen von *TS-Anwendungen* in PDN und BS2000, welche Funktionen der NEA-Transportprotokolle nutzen, die in ISO-Transportprotokollen nicht angeboten werden.

Nachricht

Eine logisch zusammengehörige Datenmenge, die an einen *Kommunikationspartner* gesendet werden soll.

NEABX

Migrationsprotokoll für den Übergang von einem NEA-Transportsystem auf ein ISO-Transportsystem.

OSI-Referenzmodell

Open Systems Interconnection ist die von der International Standards Organization ISO in der Norm ISO 7498 definierte Kommunikationsarchitektur, die einen zuverlässigen Datenaustausch zwischen Anwendungen definiert, die auf unterschiedlichen Hardware-Plattformen ablaufen. Zur Lösung dieser komplexen Aufgabe unterscheidet das OSI-Referenzmodell sieben aufeinander aufsetzende Teilaufgaben, wobei jede dieser Teilaufgaben von einer bestimmten Schicht erbracht wird. Die untereren

vier Schichten repräsentieren das *Transportsystem*, die oberen drei Schichten repräsentieren die Sicht der *Anwendung*, z. B. die Datenformate.

Passiver Partner

Ein Kommunikationspartner, der eine Verbindung nicht selbst aufbaut, sondern von einem anderen Kommunikationspartner angesprochen wird.

PDN-Anwendung

Eine *TS-Anwendung*, die in einem Kommunikationsrechner abläuft.

Prozess

Ein Prozess umfasst die Ausführung eines Programms. Er besteht aus dem ablauffähigen Programm, den Programmdaten und einer Reihe prozess-spezifischer Verwaltungsdaten, die zur Steuerung des Programms erforderlich sind.

TEP

XTI-Transportendpunkte und bei *CMX* angemeldete Prozesse bzw. Threads von *TS-Anwendungen*.

TNS (Transport Name Service)

Der TNS ist eine Komponente von *CMX*, die die korrekte Abbildung der *GLOBALEN NAMEN* von *CMX-Anwendungen* im Netz in *TRANSPORTADRESSEN* und *LOKALE NAMEN* unterstützt. Als Administrator konfigurieren Sie die von Ihnen gewählte Zuordnung von GLOBALEM NAMEN zu TRANSPORTADRESSE für ferne Anwendungen sowie die Zuordnung von GLOBALEM NAMEN zu LOKALEM NAMEN für lokale Anwendungen. Als Programmierer von Anwendungen können sie diese Abbildungen über ein *API* nutzen und damit allein mit den *GLOBALEN NAMEN* von Anwendungen ohne Bewertung der Abbilder arbeiten.

Der TNS bietet die netzweite Identifikation von Anwendungen durch logische GLOBALE NAMEN und deren Abbildung in eine entsprechende *Netzadresse*. Damit können Sie die Anwendungen vom Wissen um ihre Netzadressen entkoppeln. Zusammen mit dem *FSS* bietet der TNS die vollständige Abbildung des logischen Namens in eine konkrete *Subnetz-Adresse* und eine *Route* durch die verschiedenen Subnetze des Netzes.

TRANSPORTADRESSE einer Anwendung

Eine rufende *CMX-Anwendung* übergibt die TRANSPORTADRESSE eines gerufenen Kommunikationspartners beim Aufbau der Kommunikation an *CMX*. CMX verwendet die TRANSPORTADRESSE, um den Kommunikationspartner im Netz zu lokalisieren und eine *Route* durch das Netz zu bestimmen. Die TRANSPORTADRESSE hängt im Allgemeinen von der logischen und physischen Struktur des Netzes (und seiner Subnetze) ab. Die TRANSPORTADRESSE enthält die für Ihr Netz spezifischen Vorgaben Ihres/Ihrer Netzbetreiber(s). Als Administrator können Sie unabhängig von der Anwendung die TRANSPORTADRESSE und damit die Kommunikationswege beeinflussen.

Bestandteile einer TRANSPORTADRESSE sind: eine Netzadresse zur eindeutigen Bestimmung des fernen Rechners, auf dem die Anwendung residiert, der Typ des *Transportsystems*, über das die ferne Anwendung erreicht werden kann, und der *T-Selektor*, der die ferne Anwendung im fernen Rechner identifiziert.

Beispiele für Netzadressen sind: die Internet-Adresse in der Punktnotation "192.11.44.1", die NEA-Netzadresse in der Notation Prozessor-/Regionsnummer "47/11" und die X.25-Adresse (DTE-Adresse) als Ziffernstring "45890010123".

Als Administrator können Sie dem *GLOBALEN NAMEN* einer Anwendung eine TRANSPORTADRESSE zuordnen. Als Programmierer können Sie die TRANSPORTADRESSE mit Hilfe des Funktionsaufrufs t_getaddr() aus dem GLOBALEN NAMEN gewinnen.

Transportreferenz

Eine Nummer, die innerhalb einer *TS-Anwendung* eine *Verbindung* eindeutig kennzeichnet.

Transportsystem

Das Transportsystem bezeichnet die unteren vier Schichten des *OSI-Referenzmodells*. Ein *CCP* implementiert die vier Schichten des Transportsystems. Das Transportsystem sorgt für den gesicherten Datenaustausch zwischen Rechnern, deren *Anwendungen* miteinander kommunizieren, und zwar unabhängig von den darunterliegenden Netzstrukturen. Das Transportsystem verwendet dazu Protokolle.

TS-Anwendung

Anwendung, die die Dienste des Transportsystems nutzt. Sie besteht aus Programmen, die eine logische Verbindung zu einer anderen TS-Anwendung aufbauen können, um mit dieser Daten auszutauschen.

TSAP (Transport Service Access Point)

Zugriffspunkt für eine *TS-Anwendung* auf das Transportsystem.

T-Selektor

Der T-Selektor identifiziert eine Kommunikationsanwendung innerhalb des Rechners, auf dem die Anwendung abläuft. Der T-Selektor bildet zusammen mit der *Netzadresse* des Rechners die *TRANSPORTADRESSE* einer Anwendung, mit der diese Anwendung innerhalb eines Netzes eindeutig adressiert werden kann.

TSP (Transport Service Provider)

Ein TSP ist eine Komponente eines *CCP* oder von *CMX*, die mit Ausnahme des NTP (Null-Transport) mittels eines Transportprotokolls den OSI-Transportdienst im Netz anbietet. Sie können als Administrator die Nutzung eines bestimmten TSP für die Kommunikation von *Anwendungen* bestimmen. Der RFC1006 ist der TSP in CMX, der zusammen mit TCP/IP im Internet den OSI-Transportdienst bietet. Der NTP (Null-Transport) bietet *CMX-Anwendungen* den Direktzugriff auf die Netzdienste des X.25-Subnetzes. TP0/2, TP4 und NEA sind die TSPs für ein OSI-Umfeld und das TRANSDATA-Netz.

Ein TSP bildet zusammen mit einem *Subnetzprofil* ein *Transportsystem*. Er bietet einen Satz von konfigurierbaren Laufzeit- und Tuningparametern, bewertet die *TRANSPORTADRESSE* und findet eine geeignete Route durch das Netz. Der TSP nutzt dazu Ihre Angaben im *FSS*, soweit erforderlich.

Verbindung, logische

Zuordnung zweier Kommunikationspartner, die es ihnen ermöglicht, Daten miteinander auszutauschen.

Abkürzungen

ASCII
American Standard Code of Information Interchange

CC
Communication Controller

CCITT
Comité Consultatif International Télégraphique et Téléphonique

CCP
Communication Control Program

CMX
Communications Manager in Solaris

DCAM
Data Communication Access Method

DMA
Direct Memory Access

EBCDIC
Extended Binary Coded Decimals Interchange Code

EBNF
Extended Backus Naur Form

EMDS
Emulation Datensichtstation

EOF
End of File

EOS
End of String

ETHN
ETHERNET

ETSDU
Expedited Transport Service Data Unit

FT
File Transfer

FSB
Forwarding Support Base

ICMX
Programmschnittstelle zu CMX

IS
Intermediate System

ISDN
Integrated Services Digital Network

ISO
International Organization for Standardization

ITU
International Telecommunication Union

ITU-T
Telecommunication Standardization Sector

KD
Konfigurationsdatei

KOGS
Konfigurationsorientierte Generatorsprache

KR
Kommunikationsrechner

LAN
Local Area Network

MES
Menü-Entwicklungssystem

MSV1
Übertragungsprozedur Medium Speed Variante 1

NEA
Netzwerk-Architektur bei TRANSDATA-Systemen

NSAP
Network Service Access Point

OSI
Open Systems Interconnection

PDN
Programmsystem für Datenfernverarbeitung und Netzsteuerung

PID
Process Identifier

PVC
Permanent Virtual Circuit

QD
Quelldatei

REMOS
Remote Operation System für LAN-Kopplung

SNA
Systems Network Architecture

SNID
Subnetz-Identifikation

SNPA
Subnet Point of Access

TCEP
Transport Connection Endpoint

TCP/IP
Transmission Control Protocol/Internet Protocol

TEP
Transport Endpoint

TIDU
Transport Interface Data Unit

TLI
Transport Level Interface

TNSX
Transport Name Service in Solaris

TPDU
Transport Protocol Data Unit

TR
Token-Ring

TREF
Transport Reference

TS
Transport Service

TSAP
Transport Service Access Point

TSDU
Transport Service Data Unit

TSTAT
TEP-Status

VAR
Verarbeitungsrechner

WAN
Wide Area Network

XTI
X/OPEN Transport Interface

Literatur

Die Handbücher sind online unter *http://manuals.fujitsu-siemens.com* zu finden oder in gedruckter Form gegen gesondertes Entgelt unter *http://FSC-manual-shop.com* zu bestellen.

[1] **CMX V6.0** (Solaris)
Communications Manager UNIX
Betrieb und Administration
Benutzerhandbuch

Zielgruppe
Systemverwalter

Inhalt
Das Handbuch beschreibt den Funktionsumfang von CMX als Vermittler zwischen Anwendungen und dem Transportsystem. Es enthält Basisinformationen zur Konfigurierung und Administration von vernetzten Systemen.

[2] **CMX V5.1** (Reliant UNIX)
Communications Manager in UNIX
Betrieb und Administration
Benutzerhandbuch

Zielgruppe
Systemverwalter

Inhalt
Das Handbuch beschreibt den Funktionsumfang von CMX als Vermittler zwischen Anwendungen und dem Transportsystem. Es enthält Basisinformationen zur Konfigurierung und Administration von vernetzten Reliant UNIX-Systemen.

[3] **XTI V6.0**
X/Open Transport Interface
User Guide

Zielgruppe
Programmierer von TS-Anwendungen

Inhalt
Das Handbuch enthält implementierungsabhängige Ergänzungen zu den Funktionsaufrufen von XTI.

[4] **CMX/CCP V6.0** (Solaris)
WAN-Kommunikation
Benutzerhandbuch

Zielgruppe
Netzverwalter

Inhalt
Das Handbuch beschreibt die Rechnerkopplung über WAN (Wide Area Network); damit wird Kommunikation im Fernbereich (Wide Area Network, WAN) ermöglicht.

[5] **CCP-WAN V5.1** (Reliant UNIX)
Communication Control Program
Benutzerhandbuch

Zielgruppe
Netzverwalter

Inhalt

- Rechnerkopplung über WAN (Wide Area Network)
- WAN-Anschlüsse von CCP-WAN
- Protokoll-Profile von CCP-WAN
- Betrieb und Administration von CCP-WAN
- Konfiguration der Protokoll-Profile

[6] **CMX/CCP V6.0** (Solaris)
ISDN-Kommunikation
Benutzerhandbuch

Zielgruppe
Netzverwalter

Inhalt
Das Handbuch beschreibt die Rechnerkopplung über ISDN (Integrated Services Digital Network).

[7] **CCP-ISDN V5.1** (Reliant UNIX)
Communication Control Program
Benutzerhandbuch

Zielgruppe
Netzverwalter

Inhalt
Das Handbuch beschreibt die Rechnerkopplung über ISDN (Integrated Services Digital Network); damit wird Kommunikation im Fernbereich (Wide Area Network, WAN) ermöglicht.

[8] **CMX V6.0** (Solaris)
TCP/IP über WAN/ISDN
Benutzerhandbuch

Zielgruppe
Das Handbuch richtet sich an Netzverwalter und Systemadministratoren.

Inhalt
Das Handbuch beschreibt, wie CMX den verbindungslosen IP-Verkehr über das verbindungsorientierte WAN ermöglicht.

[9] **CS-ROUTE V2.0** (Reliant UNIX)
Benutzerhandbuch

Zielgruppe
Das Handbuch richtet sich an Netzverwalter und Systemadministratoren.

Inhalt
Beschreibung von CS-ROUTE, das den TCP/IP- OSI TP4/CLNP-Netzzugang zu X.25- und ISDN-Netzen und das parallele LAN-WAN-Routing von IP- und CLNP-Paketen über X.25 und ISDN ermöglicht.

[10] **CMX V6.0**
CS-GATE
Benutzerhandbuch

Zielgruppe
Dieses Handbuch richtet sich an Netzverwalter und Systemadministratoren.

Inhalt
Das Handbuch beschreibt, wie Sie mit CS-GATE als Gateway LAN-WAN-, LAN-LAN-, und LAN-WAN-LAN- Übergänge realisieren. Es erklärt Adressierung, Adress-Umsetzungsfunktionen und Konfigurierung sowie Steuerung und Diagnose von CS-GATE.

[11] **Reliant UNIX 5.45**
Netzverwaltung
Systemverwalterhandbuch

Zielgruppe
Systemverwalter

Inhalt
Das Handbuch beschreibt die Netzverwaltungsaufgaben, die beim Einsatz der TCP/IP-Software auf Reliant UNIX sowie der Netzbasisfunktionen (BNU) anfallen.

Stichwörter

Z

Fujitsu Siemens Computers GmbH
Handbuchredaktion
81730 München

Fax: 0700 / 372 00000

email: manuals@fujitsu-siemens.com
http://manuals.fujitsu-siemens.com

Kritik
Anregungen
Korrekturen

Absender

Kommentar zu CMX V6.0
Anwendungen programmieren

Fujitsu Siemens Computers GmbH
Handbuchredaktion
81730 München

Fax: 0700 / 372 00000

email: manuals@fujitsu-siemens.com
http://manuals.fujitsu-siemens.com

Kritik
Anregungen
Korrekturen

Absender

Kommentar zu CMX V6.0
Anwendungen programmieren

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.