EDT V17.0A Unicode-Modus

Unterprogrammschnittstellen

Ausgabe Dezember 2007

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.

Sie können uns Ihre Kommentare per E-Mail an manuals@fujitsu-siemens.com senden.

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

Inhalt

1 Einleitung

Der EDT ist der Dateieditor des BS2000, mit dem BS2000-Dateien der Formate SAM und ISAM sowie textartige Bibliothekselemente und POSIX-Dateien auf komfortable Weise erstellt und bearbeitet werden können.

Die Durchführung der beim Editieren häufig vorkommenden Arbeiten, wie z.B. Löschen, Ändern, Einfügen und Kopieren von Sätzen und Zeichen, Suchen nach Sätzen mit bestimmten Zeichenfolgen, Ausgeben von Sätzen usw. werden durch leistungsfähige und dennoch leicht erlernbare Anweisungen unterstützt.

Der EDT V17.0A kann in einem erweiterten Unicode-Modus und einem V16.6-kompatiblen Kompatibilitäts-Modus betrieben werden.

- Im Unicode-Modus kann der EDT V17.0A in Unicode und anderen Zeichensätzen codierte Dateien bearbeiten. Dem Anwender wird dabei eine komfortable Unterstützung geboten, dazu zählen u.a. die Möglichkeit, unterschiedlich codierte Dateien in verschiedenen Arbeitsdateien des EDT gleichzeitig zu bearbeiten sowie die Aufhebung der Begrenzung der Zeilenlänge (bisher 256 Zeichen). Der EDT kann beim Lesen aus und Schreiben in Dateien alle von DVS und LMS angebotenen Satzlängen verarbeiten. Bei POSIX-Dateien kann er Zeilen mit einer Maximallänge von 32768 Zeichen verarbeiten.

  Die interne Verwendung einer Unicode-Darstellung im EDT hat zur Konsequenz, dass alle Schnittstellen, an denen der Anwender bisher direkten Zugriff auf die internen Daten des EDT hatte, nicht kompatibel bleiben können. Dies trifft auf die alte L-Modus-Unterprogramm-Schnittstelle, auf die bisherige @RUN-Schnittstelle und auf den Locate-Mode der IEDTGLE-Schnittstelle zu. Diese Schnittstellen können daher im Unicode-Modus nicht mehr verwendet werden.

- Der Kompatibilitäts-Modus bietet die volle Funktionalität des EDT V16.6B mit nur geringfügigen Erweiterungen.

Obwohl der EDT als Dialogprogramm konzipiert ist, kann er auch im Stapelbetrieb Dateien und Bibliothekselemente bearbeiten.

Dateibearbeitungen, die häufig in gleicher oder ähnlicher Form auszuführen sind, lassen sich mit EDT-Prozeduren programmieren.

Der EDT kann andere Programme als Unterprogramm aufrufen und kann selbst von einem Benutzerprogramm als Unterprogramm aufgerufen werden.

1.1 Konzept der EDT-Dokumentation

In den Handbüchern

- EDT V17.0A Unicode-Modus Anweisungen
- EDT V17.0A Unicode-Modus Unterprogrammschnittstelle

wird der Unicode-Modus des EDT beschrieben. Der Kompatibilitäts-Modus wird in den Handbüchern

- EDT V16.6B Anweisungen
- EDT V16.6A Unterprogrammschnittstelle

beschrieben.

Zusätzlich enthält das Handbuch EDT V17.0A Anweisungen noch einen Abschnitt, in dem die Erweiterungen des Kompatibilitäts-Modus gegenüber dem EDT V16.6B beschrieben sind.

Die Handbücher zu den EDT-Anweisungen beschreiben grundlegende Konzepte des EDT im jeweiligen Modus und dienen als Nachschlagwerk für die zahlreichen Anweisungen des EDT.

Die Handbücher zu den Unterprogrammschnittstellen beschreiben, wie Benutzerprogramme programmiert werden können, die vom EDT aufgerufen werden können bzw. die den EDT als Unterprogramm aufrufen wollen. Sie können nur in Verbindung mit den Handbüchern zu den EDT-Anweisungen sinnvoll genutzt werden.

1.2 Zielgruppen der EDT-Handbücher

Während sich das Handbuch zu den EDT-Anweisungen an den EDT-Einsteiger und den EDT-Anwender richtet, wendet sich das Handbuch zu den EDT-Unterprogrammschnittstellen an den erfahrenen EDT-Anwender und Programmierer, der den EDT in eigene Programme einbinden will.

Dieses Handbuch zu den EDT-Unterprogrammschnittstellen richtet sich an den erfahrenen EDT-Anwender und Programmierer, der die vielfältigen Möglichkeiten des EDT in eigenen Programmen nutzen will.

Zum Aufruf des EDT als Unterprogramm sind neben der Kenntnis der wichtigsten BS2000-Kommandos, das Vertrautsein mit dem EDT und den EDT-Anweisungen, vor allem Assembler- und C-Kenntnisse unbedingte Voraussetzung.

1.3 Konzept des Handbuches EDT-Unterprogrammschnittstellen

Dieses Handbuch beschreibt ausschließlich die Unterprogrammschnittstelle des EDT V17.0A.

Dieses Handbuch enthält folgende Themen:

– **Einleitung**

  Hinweise zur Struktur und Verwendung der EDT-Handbücher.

– **Neuerungen und Änderungen in EDT V17.0A**

  Zusammenfassende Beschreibung der Neuerungen und Änderungen an der Unterprogrammschnittstelle des EDT V17.0A.

– **Nutzung des EDT als Unterprogramm**

  Beschreibung der Funktionen mit Aufruf und Returncodes, Aufbau der Kontrollblöcke, kurze Beispiele.

– **Benutzerdefinierte Anweisungen - @USE**

  Darstellung der Möglichkeit im EDT, eigene Anweisungen zu schreiben. Spezialanwendung als Anweisungsfilter.

– **Anwenderroutine - @RUN**

  Starten einer Anwenderroutine als Unterprogramm mit der Anweisung @RUN.

– **Produktion von Anwendungen der Unterprogramm-Schnittstelle**

  Regeln und Beispiele, wie im BS2000 Programme produziert werden können, die den EDT als Unterprogramm verwenden bzw. die vom EDT über die Anweisungen @USE oder @RUN als Anwenderroutinen aufgerufen werden sollen.
  Dabei werden die Sprachen C und Assembler betrachtet.

– **Ausführliche Beispielprogramme mit Kommentaren**

  Für C- und Assembler-Hauptprogramme, C- und Assembler-Anwendungsroutinen.

– **Anhang - C-Header**

  Layout-Darstellung der C-Header-Files.

Literaturhinweise werden im Text in Kurztiteln angegeben. Der vollständige Titel jeder Druckschrift, auf die durch eine Nummer verwiesen wird, ist im Literaturverzeichnis hinter der entsprechenden Nummer aufgeführt.

2 Geänderte und neue Funktionalität im EDT V17.0A

An der Unterprogrammschnittstelle ergeben sich die nachfolgend beschriebenen Neuerungen und Änderungen.

2.1 Betriebsmodi des EDT

EDT V17.0A kann in zwei Modi betrieben werden:

– Im Unicode-Modus, der für die Verarbeitung von Unicode-Dateien erweitert wurde, aber in dem einige ältere Schnittstellen nicht mehr unterstützt werden.

– Im Kompatibilitäts-Modus, der den vollen Funktionsumfang des EDT V16.6B umfasst, aber nicht die Funktionserweiterungen des Unicode-Modus bietet.

2.1.1 Unicode-Modus

Der Unicode-Modus bietet Erweiterungen für die Verarbeitung von Unicode-Dateien und unterstützt Satzlängen von mehr als 256 Byte.
Nicht unterstützt werden

– die alte L-Modus- Unterprogrammschnittstelle

– die @RUN-Schnittstelle im bisherigen Format

– der Locate-Mode der `IEDTGLE`-Schnittstelle

Der Unicode-Modus bietet eine neue, erweiterte @RUN-Schnittstelle mit geändertem Anweisungsformat und neuer Programmschnittstelle sowie Erweiterungen bei der @UNLOAD- und der @USE-Anweisung zur Unterstützung von bis zu 32 Zeichen langen Namen einer Einsprungstelle (`ENTRY`) mit Unterscheidung von Groß- und Kleinschreibung.

2.1.2 Kompatibilitäts-Modus

Im Kompatibilitäts-Modus werden der Funktionsumfang und die Schnittstellen des EDT V16.6B unterstützt. Insbesondere werden die alte L-Modus-Schnittstelle, die @RUN-Schnittstelle im bisherigen Format und der Locate-Mode der IEDTGLE-Schnittstelle in vollem Umfang unterstützt.

Die Satzlänge bleibt auf 256 Byte beschränkt.

2.1.3 Schnittstellenformate und Betriebsmodi

Für die IEDTGLE-Schnittstelle wird ein neues Format angeboten.

Damit gibt es zwei Formate:

– V16-Format: identisch mit der IEDTGLE-Schnittstelle des EDT V16.6B

– V17-Format: neues Format mit Erweiterungen für Unicode

Beim Generieren der Kontrollblöcke für die IEDTGLE-Schnittstelle kann der Benutzer die gewünschte Version wählen (siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Grundsätzlich gilt:

– Die IEDTGLE-Schnittstelle (V16-Format) wird nur vom Kompatibilitäts-Modus in vollem Umfang unterstützt

– Die IEDTGLE-Schnittstelle (V17-Format) wird nur vom Unicode-Modus in vollem Umfang unterstützt.

Um Anwendungen schreiben zu können, die die IEDTGLE-Schnittstelle nutzen und sowohl mit dem Kompatibilitäts-Modus als auch mit dem Unicode-Modus laufen können, gibt es einen neuen Verbindungsmodul, der die Schnittstellen umsetzt. Damit ist es auch möglich, dass eine Anwendung, die mit der IEDTGLE-Schnittstelle des EDT V16.6B arbeitet, ohne Umstellung auch im Unicode-Modus des EDT V17.0A läuft, wenn sie nur solche Funktionen nutzt, die auch der Unicode-Modus unterstützt.

Falls noch kein EDT läuft, oder bei laufendem EDT alle Arbeitsdateien leer sind, wird bei Aufruf der IEDTCMD-Schnittstelle der passende Betriebsmodus aktiviert: Kompatibilitäts-Modus für V16-Format, Unicode-Modus für V17-Format. Sobald eine explizite Umschaltung des Betriebsmodus mit der @MODE- oder @CODENAME-Anweisung vorgenommen wurde, tritt dieser Automatismus außer Kraft. Nachfolgende Moduswechsel müssen dann immer explizit vorgenommen werden. Ebenso findet niemals ein impliziter Moduswechsel aus einer Anwenderroutine heraus statt. Andernfalls ginge der Rücksprung aus der Anwenderroutine in den EDT ins Leere.

Man beachte, dass ein implizites Umschalten des Betriebsmodus durch Wechsel der Schnittstellenversion nur über die IEDTCMD-Schnittstelle möglich ist. Da der EDT sich nach dem Umschalten im nicht initialisierten Zustand befindet, würden alle Funktionsaufrufe, die einen initialisierten EDT voraussetzen, zum Fehler führen.

Generell empfiehlt sich, innerhalb eines Programms stets mit dem gleichen Format der Schnittstelle (V17-Format oder V16-Format) zu arbeiten und eventuell notwendige Wechsel des Betriebsmodus explizit (via @MODE-Anweisung) vorzunehmen.

Falls der EDT bereits läuft und ein Wechsel des Betriebsmodus nicht möglich ist oder aufgrund eines vorangegangenen expliziten Wechsels nicht erfolgt (s.o.), wird das Schnittstellenformat umgesetzt:

– Ein Aufruf der IEDTGLE-Schnittstelle im V16-Format wird auf V17-Format umgesetzt, wenn der EDT im Unicode-Modus läuft.

– Ein Aufruf der IEDTGLE-Schnittstelle im V17-Format wird auf V16-Format umgesetzt, wenn der EDT im Kompatibilitäts-Modus läuft.

Diese Umsetzung ist nur möglich, wenn die genutzte Funktion vom gerade laufenden Betriebsmodus des EDT unterstützt wird. Beispielsweise kann das V16-Format des AMCB nicht in das V17-Format umgesetzt werden, wenn Locate-Mode verlangt wird. In diesem Fall wird der Returncode EAMPAERR/EAMPA08 geliefert.

Bei Nutzung der IEDTGLE-Schnittstelle im V16-Format gilt:

– Der Locate-Mode kann nicht umgesetzt werden.

– Beim Lesen des globalen Status (Funktion IEDTGET mit Pseudo-Arbeitsdatei 'G'), wird das im V17-Format entfallene Feld für den global eingestellten Zeichensatz nur dann mit einem Wert ungleich Leerzeichen versorgt, wenn in allen nicht leeren Arbeitsdateien der gleiche Zeichensatz eingestellt ist.

Für die IEDTGLE-Schnittstelle im V17-Format ist festgelegt, welche Teilmenge ins V16-Format umgesetzt werden kann (*kompatibles* V17-Format). Dies wird bei der Beschreibung jeweils angegeben.

Der volle Umfang des V17-Formats (*erweitertes* V17-Format) kann nur mit dem Unicode-Modus verwendet werden.

Um sicherzustellen, dass nicht versehentlich Erweiterungen benutzt werden, kann an der Schnittstelle im V17-Format angegeben werden, dass nur das kompatible Format genutzt werden soll (durch Setzen des Flag EGLCOMP). In diesem Fall wird die Nutzung von Funktionen des erweiterten Formates mit Returncode abgewiesen (auch im Unicode-Modus).

Bei Verwendung des erweiterten Formats (Flag EGLCOMP nicht gesetzt) wird zudem verhindert, dass eine automatische Umsetzung des V17-Formats in das V16-Format stattfindet, auch wenn dies möglich wäre. Man beachte aber, dass das Flag ein *Umsetz*-Flag und kein *Umschalt*-Flag ist. D.h. auch mit kompatiblem Format findet ggf. ein *Umschalten* in den Unicode-Modus statt, wenn dies möglich ist.

Das V16-Format ist identisch mit dem Format der IEDTGLE-Schnittstelle des EDT V16.6B. Daher kann eine Anwendung, die das *kompatible* V17-Format nutzt, auch mit EDT V16.6B laufen, wenn sie den Verbindungsmodul IEDTGLE des EDT V17.0A eingebunden hat.

Die Übersicht zeigt, welche Kombinationen möglich sind:

	EDT V17 nicht vorhanden	**EDT V17 vorhanden-Umschalten erlaubt und Aufruf über IEDTCMD**	**EDT V17 vorhanden-Kompatibilitätsmodus aktiv, Umschalten verboten**	**EDT V17 vorhanden-Unicode-Modus aktiv, Umschalten verboten**
Aufruf über IEDTGLE V16 Schnittstelle	Wie bisher	Kompatibilitäts-Modus einstellen	Keine Sonderbehandlung nötig	Umsetzen auf V17 Schnittstelle-wenn nicht möglich: Fehler
Aufruf über IEDTGLE V17 Schnittstelle (kompatibles Format)	Umsetzen auf V16 Schnittstelle	Unicode-Modus einstellen	Umsetzen auf V16 Schnittstelle	Keine Sonderbehandlung nötig
Aufruf über IEDTGLE V17 Schnittstelle (erweitertes Format)	Fehler	Unicode-Modus einstellen	Fehler	Keine Sonderbehandlung nötig
Aufruf über L-Modus Schnittstelle	Wie bisher	Kompatibilitäts-Modus einstellen	Keine Sonderbehandlung nötig	Fehler

2.2 Lange Sätze

Im Unicode-Modus können Sätze bis zu einer Länge von 32768 **Zeichen** bearbeitet werden. An der IEDTGLE-Schnittstelle können Sätze aber nur bis zu einer Länge von 32768 **Bytes** (wie beim DVS) übertragen werden.

Dazu ist keine Erweiterung der IEDTGLE-Schnittstelle erforderlich. Dies gilt auch für die IEDTGLE-Schnittstelle im V16-Format. Für die ausreichende Länge der Puffer hat der Anwender zu sorgen.

2.3 Lokale Zeichensätze

Im Unicode-Modus können für die einzelnen Arbeitsbereiche verschiedene Zeichensätze eingestellt werden. Daher wird die Zeichensatz-Information nicht mehr beim Lesen der globalen Einstellungen übertragen, sondern beim Lesen der arbeitsdateispezifischen Einstellungen; das entsprechende Feld befindet sich nicht mehr im Kontrollblock `EDTPARG` sondern im Kontrollblock `EDTPARL`.

2.4 Locate-Mode

Der Locate-Mode der `IEDTGLE`-Schnittstelle wird im Unicode-Modus des EDT V17.0A nicht unterstützt.

2.5 Speicherorganisation

Im Unicode-Modus des EDT V17.0A hat der Benutzer keinen Einfluss auf die Speicherreorganisation des EDT. Dies ist nicht mehr notwendig, da der Locate-Mode nicht unterstützt wird und der Benutzer keinen direkten Zugriff mehr auf Sätze im EDT-Speicher hat.

2.6 L-Modus-Schnittstelle

Die alte L-Modus-Schnittstelle (Vorgänger der `IEDTGLE`-Schnittstelle) wird im Unicode-Modus des EDT V17.0A nicht unterstützt.

2.7 @RUN-Schnittstelle

Der Unicode-Modus des EDT V17.0A bietet eine neue @RUN-Schnittstelle mit geändertem Anweisungsformat und neuer, erweiterter Programmschnittstelle.

2.8 @UNLOAD-Anweisung

Die @UNLOAD Anweisung wurde im Unicode-Modus um Operanden erweitert, die das Entladen einer kompletten Ladeeinheit des BLS (`UNIT`) gestatten.

2.9 @USE-Anweisung

Die @USE-Anweisung wurde im Unicode-Modus um Operanden erweitert, die es gestatten, als Einsprungpunkt (ENTRY) einen bis zu 32 Zeichen langen Namen mit Unterscheidung von Groß- und Kleinschreibung anzugeben.

2.10 Benutzerdefinierte Anweisungen und Anwenderroutinen in Hochsprachen

Benutzerdefinierte Anweisungen, die über die @USE-Schnittstelle vereinbart werden und Anwenderroutinen, die über die @RUN-Schnittstelle aufgerufen werden, können in Hochsprachen (z.B. C) geschrieben werden, sofern diese Sprachen die ILCS-Linkage unterstützen. Gegebenenfalls muss die ILCS-Linkage via Compiler-Option aktiviert werden.

2.11 Leere Sätze

EDT V17.0A kann im Unicode-Modus Sätze der Satzlänge 0 lesen und schreiben.

Auch bei Nutzung des V16-Formats können Sätze der Länge 0 gelesen und geschrieben werden, wenn der EDT im Unicode-Modus läuft.

2.12 Schnittstellen und ihr Zusammenspiel

Der EDT bietet drei Schnittstellen:

- Eine Schnittstelle, bei der ein Anwenderprogramm Funktionen des EDT ruft (IEDTGLE-Schnittstelle)
- Eine Schnittstelle, bei der eine vom Anwender geschriebene Routine als benutzerdefinierte Anweisung vereinbart wird. Zur Ausführung einer solchen benutzerdefinierten Anweisung wird die zugehörige Routine gerufen (@USE-Schnittstelle).
- Eine Schnittstelle zum Aufruf einer Anwenderroutine (@RUN)

Diese drei Schnittstellen verwenden das gleiche Layout der Kontrollblöcke.

Mit der @USE-Schnittstelle kann auch ein Anweisungsfilter vereinbart werden:
die vereinbarte Routine wird bei jeder eingegebenen Anweisung gerufen, um zu entscheiden, ob diese Anweisung ausgeführt werden soll oder nicht.

Eine Routine, die eine benutzerdefinierte Anweisung realisiert, und eine Anwenderroutine können ihrerseits über die IEDTGLE-Schnittstelle Funktionen des EDT aufrufen (teilweise mit Einschränkungen).

3 Nutzung des EDT als Unterprogramm

Die Aufruf-Schnittstelle des EDT besteht aus folgenden Teilen:

- einem Verbindungsmodul (IEDTGLE) mit mehreren Einsprungadressen und
- einer Reihe von Kontrollblöcken, die mit Assembler-Makros oder C-Includes generiert werden.

Beim Aufruf einer Funktion wird die entsprechende Einsprungadresse gerufen und es werden ihr die Adressen der benötigten Kontrollblöcke und Puffer als Parameter mitgegeben. Anzahl und Art der Parameter ist für jede Funktion unterschiedlich. Für jede Funktion ist festgelegt, welche Felder der übergebenen Kontrollblöcke ausgewertet und versorgt werden (siehe Abschnitt „Anweisungsfunktionen“ auf Seite 52 und „Logische Satzzugriffsfunktionen“ auf Seite 66).

3.1 Verknüpfung des Benutzerprogramms mit dem EDT

Im Benutzerprogramm (Hauptprogramm) wird über eine Externreferenz (z.B eine V-Konstante) der Modul IEDTGLE aus der Modulbibliothek dazugebunden.

Der Modul IEDTGLE

- enthält alle Einsprungadressen für die Anwendung der einzelnen Funktionen
- ruft mit Hilfe des BIND-Makros den nachladbaren bzw. vorgeladenen Teil des EDT auf
- sichert die Einsprungadresse des EDT im globalen Kontrollblock EDTGLCB

Das BIND-Makro wird nur beim Erstaufruf durchlaufen. Weitere Aufrufe entnehmen die Einsprungadresse dem Kontrollblock EDTGLCB.

Der Modul IEDTGLE ist reentrant geschrieben. Über IEDTGLE wird in den EDT verzweigt. Falls der Aufruf mit dem kompatiblen V17-Schnittstellenformat erfolgt und im aktuellen System kein V17-EDT installiert ist, setzt bereits der IEDTGLE das V17-Format (falls möglich und erlaubt) in das V16-Format um, ansonsten werden die Parameter unverändert an den EDT weitergegeben.

3.1.1 Aufruf des EDT

Der EDT wird nach den Standard-Programmverknüpfungsregeln aufgerufen. Er kann auch von höheren Programmiersprachen aufgerufen werden. Beim Sprung in den EDT müssen die Register wie folgt geladen sein:

Register	Datenbereich
(R1)	A (PARAMETERLISTE)
(R13)	A (SAVEAREA)
(R14)	A (RETURN)
(R15)	V (ENTRY)

PARAMETERLISTE

Der Benutzer muss diesen Datenbereich selbst erstellen.

Die Parameterliste muss alle Adressen der Kontrollblöcke und definierten Datenfelder enthalten, aus denen der EDT die notwendigen Daten entnehmen kann (z.B. Anweisungsfolgen, Meldungstexte etc).

Die Parameterliste ist abhängig von der Funktion des Aufrufs. Welche Parameter anzugeben sind, ist den Beschreibungen der einzelnen Funktionen zu entnehmen (siehe Abschnitt „Anweisungsfunktionen“ auf Seite 52 und „Logische Satzzugriffsfunktionen“ auf Seite 66).

SAVEAREA

Register-Sicherstellungsbereich (18 Worte, `DC 18F'0'`), der vom Aufrufer erstellt werden muss. Der EDT sichert dort die Register.

RETURN

Rücksprungadresse im rufenden Programm. Nach Beenden der EDT-Funktionen wird das Programm an dieser Adresse fortgesetzt.

ENTRY

Der Modul `IEDTGLE` enthält für jede Funktion eine eigene Einsprungadresse:

Einsprungadresse	Funktion
`IEDTINF`	Lesen der Versionsnummer des EDT
`IEDTCMD`	Ausführen einer EDT-Anweisungsfolge
`IEDTEXE`	Ausführen einer EDT-Anweisungsfolge
`IEDTGET`	Lesen eines Satzes Lesen des globalen Status Lesen des lokalen Status einer Arbeitsdatei

Einsprungadresse	**Funktion**
IEDTGTM	Lesen eines markierten Satzes
IEDTPUT	Schreiben eines Satzes
IEDTPTM	Markieren eines Satzes
IEDTDEL	Löschen eines Satzbereiches Löschen des Kopierpuffers
IEDTREN	Ändern der Zeilennummer

3.2 Generierung und Aufbau der Kontrollblöcke

Im Folgenden werden die Makros zum Generieren der Kontrollblöcke, der Aufbau der Kontrollblöcke und die Bedeutung der Felder der Kontrollblöcke beschrieben.

Die Beschreibung erfolgt anhand der Struktur der entsprechenden Assembler-Makros. Das Layout der Schnittstellen in C (Header-Files) ist im Abschnitt „Anhang - C-Header“ auf Seite 159 beschrieben.

Felder, die nicht beschrieben werden, sind für die interne Nutzung durch den EDT vorgesehen. Der Anwender darf ihren Inhalt nicht verändern.

Die Zuordnung der Kontrollblock-Versionen zum V16- bzw. V17-Format zeigt die folgende Übersicht:

Kontollblock	**V16-Format**	**V17-Format**
EDTGLCB	Version 1	Version 2
EDTUPCB	Version 2	Version 3
EDTAMCB	Version 1	Version 2
EDTPARG	Version 1	Version 2
EDTPARL	Versionen 1, 2, 3	Version 4

Bei einem Aufruf müssen die verwendeten Kontrollblöcke entweder alle zum V16-Format oder alle zum V17-Format gehören. Werden in einem Aufruf Kontrollblöcke im V16- und V17-Format gemischt, wird der Aufruf abgewiesen.

3.2.1 EDTGLCB - Globaler EDT - Kontrollblock

Der EDTGLCB stellt den globalen Kontrollblock innerhalb aller EDT-Programmschnittstellen dar. Er wird sowohl von der IEDTGLE-Schnittstelle als auch von der @USE- und der @RUN-Schnittstelle verwendet.

Erstellen des Kontrollblockes EDTGLCB

Mit dem Assembler-Makro IEDTGLCB kann der Kontrollblock EDTGLCB generiert werden.

Name	Operation	Operanden
[name]	IEDTGLCB	[{ D / C }][,prefix] [,VERSION = { 1 / 2 }]

name – Symbolischer Name der 1. DS-Anweisung bei Angabe von C.
– Name der DSECT bei Angabe von D.

Wird *name* nicht angegeben, wird EDTGLCB benutzt (mit vorangestelltem *prefix*, falls angegeben).

D Es wird ein Pseudoabschnitt (DSECT) generiert.

C Es wird ein Speicherabschnitt mit symbolischen Adressen generiert (keine CSECT-Anweisung).

prefix Ein Zeichen, mit dem die generierten Feldnamen beginnen sollen.

Wird *prefix* nicht angegeben, wird standardmäßig E eingesetzt.

VERSION Auswahl, welche Version des Kontrollblocks generiert werden soll:

Die Version 1 wird mit dem V16-Format der Schnittstelle eingesetzt.
Die Version 2 wird mit dem V17-Format der Schnittstelle eingesetzt.

Bei Angabe des Makros `IEDTGLCB VERSION=2` wird der Kontrollblock `EDTGLCB` in folgender Form generiert:

```
           IEDTGLCB D,VERSION=2
1 EDTGLCB  MFPRE DNAME=EDT,MF=D
2 EDTGLCB  DSECT ,
2               *,##### PREFIX=I, MACID= #####
1 *--------------- EDT UNIT NUMBER, EDTGLCB VERSION NUMBER --------------
1 EGLUNITC EQU   66                  EDT UNIT NUMBER
1 EGLVERSC EQU   2                   EDTGLCB VERSION NUMBER
1 EGLVERSL EQU   12                  VERSION-LENGTH (INFO)
1 EGLMSGM  EQU   80                  MAX LENGTH FOR MESS
1 *---------------- EDT MAIN-RETURNCODES ---------------------------------
1 *                     *------ EDT-CALL ---------------------------------
1 EUPRETOK EQU   X'0000'             NO ERROR
1 EUPSYERR EQU   X'0008'             SYNTAX ERROR IN COMMAND
1 EUPRTERR EQU   X'000C'             RUNTIME ERROR IN COMMAND
1 EUPEDERR EQU   X'0010'             UNRECOVERABLE EDT ERROR
1 EUPOSERR EQU   X'0014'             UNRECOVERABLE SYSTEM ERROR
1 EUPUSERR EQU   X'0018'             UNRECOVERABLE USER ERROR
1 EUPPAERR EQU   X'0020'             PARAMETER ERROR
1 EUPSPERR EQU   X'0024'             REQM ERROR
1 EUPVEERR EQU   X'0028'             VERSION ERROR                  V16.5
1 EUPABERR EQU   X'002C'             ABNORMAL HALT BY USER          V16.5
1 EUPCMPER EQU   X'0030'             COMPATIBILITY VIOLATION        V17.0
1 *                     *------ EDT-ACCESS-METHOD ----------------------
1 EAMRETOK EQU   X'0000'             NO ERROR
1 EAMACERR EQU   X'0004'             ACCESS ERROR
1 EAMEDERR EQU   X'0010'             UNRECOVERABLE EDT ERROR
1 EAMOSERR EQU   X'0014'             UNRECOVERABLE SYSTEM ERROR
1 EAMUSERR EQU   X'0018'             UNRECOVERABLE USER ERROR
1 EAMPAERR EQU   X'0020'             PARAMETER ERROR
1 EAMSPERR EQU   X'0024'             REQM ERROR
1 *---------------- EDT SUB-RETURNCODE1 ----------------------------------
1 *                     *------ MAIN: EUPRETOK --------------------------
1 EUPOK00  EQU   X'00'               NO ERROR
1 EUPOK04  EQU   X'04'               HALT
1 EUPOK08  EQU   X'08'               HALT <TEXT>
1 EUPOK12  EQU   X'0C'               RETURN
1 EUPOK16  EQU   X'10'               RETURN <TEXT>
1 EUPOK20  EQU   X'14'               K1-KEY
1 EUPOK24  EQU   X'18'               IGNORE COMMAND
1 *                     *------ MAIN: EUPPAERR --------------------------
1 EUPPA04  EQU   X'04'               ERROR IN EDTGLCB
1 EUPPA08  EQU   X'08'               ERROR IN EDTUPCB
1 EUPPA12  EQU   X'0C'               ERROR IN COMMAND PARAMETER
1 EUPPA16  EQU   X'10'               ERROR IN MESSAGE PARAMETER
```

```
1 EUPPA20  EQU   X'14'              ERROR IN CCSN                  V17.0
1 EUPPA24  EQU   X'18'              CONVERSION ERROR               V17.0
1 *                    *------ MAIN: EUPVEERR --------------------------
1 EUPVE00  EQU   X'00'              STANDARD VERSION RETURNED
1 EUPVE04  EQU   X'04'              NO VERSION RETURNED
1 *                    *------ MAIN: EAMRETOK --------------------------
1 EAMOK00  EQU   X'00'              NO ERROR
1 EAMOK04  EQU   X'04'              NEXT RECORD RETURNED
1 EAMOK08  EQU   X'08'              FIRST RECORD RETURNED
1 EAMOK12  EQU   X'0C'              LAST RECORD RETURNED
1 EAMOK16  EQU   X'10'              FILE CLEARED
1 EAMOK20  EQU   X'14'              COPY BUFFER CLEARED            V16.6
1 *                    *------ MAIN: EAMACERR --------------------------
1 EAMAC04  EQU   X'04'              PUT RECORD TRUNCATED
1 EAMAC08  EQU   X'08'              KEY TRUNCATED ( MOVE MODE )
1 EAMAC12  EQU   X'0C'              RECORD TRUNCATED (MOVE MODE )
1 EAMAC16  EQU   X'10'              FILE IS EMPTY
1 EAMAC20  EQU   X'14'              NO MARKS IN FILE
1 EAMAC24  EQU   X'18'              FILE NOT OPENED (NO LONGER USED)
1 EAMAC28  EQU   X'1C'              FILE REAL OPENED (NO MARKS)
1 EAMAC32  EQU   X'20'              PTM NOT FOUND
1 EAMAC36  EQU   X'24'              REN KEY ERROR
1 EAMAC40  EQU   X'28'              MAX LINE ERROR
1 EAMAC44  EQU   X'2C'              RENUMBER INHIBITED
1 EAMAC48  EQU   X'30'              FILE IS ACTIVE
1 *                    *------ MAIN: EAMPAERR --------------------------
1 EAMPA04  EQU   X'04'              ERROR IN EDTGLCB
1 EAMPA08  EQU   X'08'              ERROR IN EDTAMCB
1 EAMPA12  EQU   X'0C'              FILENAME ERROR
1 EAMPA16  EQU   X'10'              ACCESS FUNCTION ERROR
1 EAMPA20  EQU   X'14'              KEY FORMAT ERROR
1 EAMPA24  EQU   X'18'              KEY LENGTH ERROR
1 EAMPA28  EQU   X'1C'              RECORD LENGTH ERROR
1 EAMPA32  EQU   X'20'              WRONG TRANSFER MODUS BYTE
1 EAMPA36  EQU   X'24'              WRONG VERSION OR UNIT NUMBER
1 EAMPA40  EQU   X'28'              ERROR IN CCSN                  V17.0
1 EAMPA44  EQU   X'2C'              CONVERSION ERROR               V17.0
1 *--------------- CONTROL BLOCK EDTGLCB ---------------------------------
1 *                      *---- CONTROL BLOCK HEADER --------------------
1 EGLFHE   DS    0XL8               GENERAL OPERAND LIST HEADER
1 EGLIFID  DS    0A                 INTERFACE IDENTIFIER
1 EGLUNIT  DC    AL2(XGLUNITC)      UNIT NUMBER
1          DS    AL1                RESERVED
1 EGLVERS  DC    AL1(XGLVERSC)      FUNCTION INTERFACE VERSION NUMBER
1 *                      *--- RETURN CODE ------------------------------
1 EGLRETC  DS    0A                 GENERAL RETURN CODE
1 EGLSRET  DS    0AL2               SUB RETURN CODE
1 EGLSR2   DC    AL1(0)             SUB RETURN CODE2
```

```
1 EGLSR1   DC    AL1(0)              SUB RETURN CODE1
1 EGLMRET  DC    AL2(0)              MAIN RETURN CODE
1 *                        *---- RETURN MESSAGE FIELD --------------------
1 EGLINFM  DS    0F                  INFORMATION OF MEMORY SIZE
1 EGLCMDS  DC    F'0'                DISPLACEMENT OF INVALID COMMAND
1 EGLRMSG  DS    0CL82               EDT RETURN MESSAGE
1 EGLRMSGL DC    H'0'                MESSAGE LENGTH
1 EGLRMSGF DC    CL80' '             MESSAGE FIELD
1 *                        *---- EDT GLOBAL PARAMETERS -------------------
1 EGLCDS   DC    X'00'               CODE OF SENDING KEY            V16.4
1 EGLDUE   EQU   X'66'               DUE
1 EGLF1    EQU   X'5B'               F1
1 EGLF2    EQU   X'5C'               F2
1 EGLF3    EQU   X'5D'               F3
1 EGLK1    EQU   X'53'               K1
1 EGLINDB  DC    X'00'               INDICATOR BYTE
1 EGLCOMP  EQU   X'80'               COMPATIBLE/EXTENDED FORMAT     V17.0
1 EGLILCS  EQU   X'40'               ILCS ENVIRONMENT               V17.0
1 EGLSPL   EQU   X'10'               EDT CALL FROM SPL
1 EGLSTXIT EQU   X'08'               EDT STXIT ALLOWED              V16.4
1 EGLINIT  EQU   X'04'               EDT DATA INITIATED
1 EGLDTVD  EQU   X'02'               EDT DATA ADDRESS VALID
1 EGLETVD  EQU   X'01'               EDT ENTRY ADDRESS VALID
1 EGLENTRY DC    A(0)                EDT ENTRY ADDRESS
1 EGLDATA  DC    A(0)                EDT DATA ADDRESS
1 *                        *---- ACTIVE EDT-FILE (OUTPUT)-----------------
1 EGLFILE  DC    CL8' '              INTERN FILENAME
1 *                        *---- USER PARAMETERS ------------------------
1 EGLUSR1  DC    XL4'00000000'       USER PARAMETER1 (EDT CALLER    )
1 EGLUSR2  DC    XL4'40404040'       USER PARAMETER2 (SUBROUTINE    )
1 EGLUSR3  DC    XL4'40404040'       USER PARAMETER3 (EXIT ROUTINE  )
1 *                        *---- CHARACTER SET ---------------------------
1 EGLCCSN  DC    CL8' '              CODED CHARACTER SET NAME       V17.0
1 EGLIND2  DC    X'00'               INDICATOR BYTE 2               V17.0
1 EGLCMOD  EQU   X'80'               COMP MODE RUNNING              V17.0
1 EGLRES   DC    X'000000'           RESERVED                       V17.0
1 *--------------- LENGTH OF CONTROL BLOCK -------------------------------
1 EGLGLCBL EQU   *-EDTGLCB
```

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EGLUNIT	Eindeutige Identifikation des EDT.	2	X	A(M)	
EGLVERS	Änderungsstand des Kontrollblocks.	1	X	A(M)	
EGLSR1	SUBCODE1: Unterwert des Returncodes, der innerhalb des Hauptwerts eindeutig ist.	1	X		R
EGLSR2	SUBCODE2: Unterwert des Returncodes, der innerhalb des Hauptwerts eindeutig ist.	1	X		R
EGLMRET	MAINCODE: Hauptwert des Returncodes. Die einzelnen Returncodes sind auf Seite 24ff näher erläutert.	2	X		R
EGLCMDS/ EGLINFM	In diesem Feld steht bei einem Anweisungsfehler (Syntaxfehler) die Distanz der fehlerhaften Anweisung zum Beginn der Anweisungsfolge (Funktionen IEDTCMD und IEDTEXE). Bei der Funktion IEDTINF wird das Feld EGLINFM vom EDT immer mit 0 belegt.	4	X		R
EGLRMSGL	Länge der Meldung in EGLRMSGF. Wird keine Meldung übergeben, enthält das Feld den Wert Null.	2	X	A	R
EGLRMSGF	In diesem Feld wird vom EDT eine (Fehler-) Meldung an das Benutzerprogramm übergeben.	80	C	A	R
EGLCDS	In diesem Feld wird einem Anweisungsfilter vom EDT die Sendetaste mitgeteilt, mit der die Anweisung im F-Modus-Dialog abgeschickt wurde. EGLDUE DUE-Taste EGLF1 Funktionstaste F1 EGLF2 Funktionstaste F2 EGLF3 Funktionstaste F3 EGLK1 Funktionstaste K1 Werte für andere Funktionstasten siehe [18].	1	X		R
EGLINDB	Das Indikator-Byte enthält einzelne Flags mit verschiedener Bedeutung.	1			

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
Flag EGLCOMP	Der Aufruf der Funktion soll auf das kompatible Format beschränkt sein: *nicht gesetzt* (Standard): Es wird das erweiterte V17-Format benutzt. *gesetzt*: Es wird das kompatible V17-Format benutzt. Siehe auch Abschnitt 2.1.3.			A	
Flag EGLILCS	Dieses Flag (*ILCS*-Flag) sollte gesetzt werden, wenn der EDT aus einem C-Hauptprogramm heraus aufgerufen wird (siehe ab Abschnitt 6.1). Wird der EDT V16 mit dem kompatiblen V17-Format aufgerufen, wird dieses Flag ignoriert.			A	
Flag EGLSPL	Dieses Flag (*SPL*-Flag) muss gesetzt werden, wenn der EDT aus einer SPL-Umgebung aufgerufen wird.			A	
Flag EGLSTXIT	Dieses Flag muss gesetzt werden, wenn die Unterbrechungsroutinen des EDT während des Aufrufs aktiviert werden sollen. Es wird nur bei Aufruf über die Schnittstellen IEDTCMD und IEDTEXE ausgewertet.			A	
Flag EGLINIT	Dieses Flag wird vom EDT nach der Initialisierung gesetzt und nach Beenden des EDT gelöscht.			L(M)	R
Flag EGLDTVD	Dieses Flag wird vom EDT nach der Initialisierung gesetzt und nach Beenden des EDT gelöscht.			L(M)	
Flag EGLETVD	Dieses Flag wird vom EDT nach der Initialisierung gesetzt. Es bleibt auch nach Beendigung des EDT gesetzt.			L(M)	
EGLFILE	Dieses Feld wird vom EDT bei Rückkehr aus der Funktion IEDTCMD linksbündig mit der Nummer der aktuellen Arbeitsdatei ($0..$22) versorgt. Das Feld wird mit Leerzeichen aufgefüllt. Dies gilt ebenfalls für den EDTGLCB, der einer Anweisungs- oder Anwenderroutine mitgegeben wird.	8	C		R

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EGLUSR1	In diesem Feld kann ein Aufrufer der IEDTCMD -Schnittstelle einen Wert hinterlegen, der vom EDT anderen Nutzern der IEDTGLE- oder der @USE-Schnittstelle unverändert zur Verfügung gestellt wird (Details siehe unten). Das Feld ist mit binär Null vorbelegt.	4		A	R
EGLUSR2	In diesem Feld kann ein Nutzer der @USE-Schnittstelle einen Wert hinterlegen, der vom EDT anderen Nutzern der @USE- oder der IEDTGLE-Schnittstelle unverändert zur Verfügung gestellt wird (Details siehe unten). Das Feld ist mit vier Leerzeichen vorbelegt.	4		A	R
EGLUSR3	Dieses Feld wird vom EDT derzeit nicht übernommen. Der Inhalt ist daher undefiniert.	4			
EGLCCSN	Name des Zeichensatzes (ggf. mit Leerzeichen aufgefüllt oder mit X'00' abgeschlossen), in dem bestimmte Parameter codiert sind: COMMAND, MESSAGE1, MESSAGE2 und EDTREC (beim Lesen und Schreiben). Siehe Abschnitte Anweisungs- und Satzzugriffsfunktionen. Wird kein Zeichensatz angegeben (acht Leerzeichen oder 1.Zeichen binär Null), wird der Zeichensatz nach den unten beschriebenen Regeln bestimmt.	8	C	A	
EGLIND2	Dieses Feld enthält momentan nur das Flag EGLCMOD.	1			
Flag EGLCMOD	Dieses Flag wird vom EDT gesetzt, wenn der EDT im Kompatibilitäts-Modus läuft, also die Kontrollblöcke aus dem V16-Format rückkonvertiert wurden.				R

A	Aufrufparameter	Muss vom Aufrufer versorgt werden.
(M)		Wird vom Makro (bei Angabe des Parameters C) gesetzt und sollte vom Anwender nicht verändert werden.
L	Aufrufparameter	Muss vom Aufrufer beim ersten Aufruf gelöscht werden und darf danach nicht mehr geändert werden.
R	Rückkehrparameter	Wird von EDT versorgt.
X	Binärformat	Binäre Zahlen.
C	Abdruckbar	Abdruckbare Texte (im Zeichensatz EDF03IRV).

Auswahl des Zeichensatzes bei fehlender Angabe in EGLCCSN

Bei den Funktionen IEDTCMD und IEDTEXE:

1. Wenn im EDT ein globaler Zeichensatz eingestellt ist, d.h. für alle nicht leeren Arbeitsdateien des EDT der gleiche Zeichensatz eingestellt ist und wenn dieser Zeichensatz ein 7-Bit oder 8-Bit EBCDIC Zeichensatz ist, wird dieser verwendet.
2. Wenn aufgrund von 1) kein Zeichensatz bestimmt werden kann, wird der Zeichensatz der aktuellen Arbeitsdatei verwendet, sofern dieser ein 7- oder 8-Bit EBCDIC Zeichensatz ist.
3. Wenn weder durch 1) noch durch 2) ein Zeichensatz bestimmt werden kann, wird der Zeichensatz EDF041 verwendet.

Bei den Funktionen IEDTGET, IEDTGTM und IEDTPUT:

1. Wenn die Arbeitsdatei, aus der gelesen bzw. in die geschrieben wird, nicht leer ist, wird der Zeichensatz dieser Arbeitsdatei gewählt.
2. Wenn die Arbeitsdatei leer ist, wird wie bei IEDTCMD und IEDTEXE verfahren.

Behandlung der Benutzer-Parameter

Der EDT verwaltet globale Instanzen der Benutzer-Parameter EGLUSR1, EGLUSR2 und EGLUSR3. Er übernimmt Werte aus einem vom Benutzer versorgten GLCB in diese globalen Instanzen in folgenden Fällen:

1. Der Wert von EGLUSR1 wird übernommen, wenn aus einem Benutzerprogramm die Schnittstelle IEDTCMD aufgerufen wird.
2. Der Wert von EGLUSR2 wird übernommen, wenn aus einer Anwenderroutine (@RUN) oder einer Anweisungs- bzw. Filterroutine (@USE) zum EDT zurückgekehrt wird. Dies gilt ebenso bei Rückkehr aus den zugehörigen Initialisierungsroutinen.

3. Der Wert von EGLUSR3 wird derzeit niemals übernommen.

Der EDT versorgt einen GLCB mit den Werten aller drei globalen Instanzen in folgenden Fällen:

1. Der EDT ruft eine Anwenderroutine (@RUN) oder eine Anweisungs- bzw. Filterroutine (@USE). Dies gilt ebenso beim Aufruf der zugehörigen Initialisierungsroutinen.
2. Der EDT gkehrt nach einem IEDTCMD- oder IEDTEXE-Aufruf in ein Anwenderprogramm zurück.

Änderungen gegenüber dem V16-Format

– Das Flag EGLREOR entfällt.
– Das Flag EGLCOMP ist neu.
– Das Flag EGLILCS ist neu.
– Das Feld EGLCCSN ist neu.
– Das Feld EGLIND2 mit dem Flag EGLMOD ist neu.
– Die Returncodes EUPCMPER, EUPPA20, EUPPA24, EAMPA40 und EAMPA44 sind neu.

Kompatibles V17-Format: das Feld EGLCCSN darf nur Leerzeichen enthalten.

3.2.2 EDTUPCB - Unterprogramm - Kontrollblock

Der EDTUPCB (Unterprogramm-Kontrollblock) enthält die Parameter, die die Voreinstellwerte des EDT bei der Funktion IEDTCMD festlegen.

Erstellen des Kontrollblockes EDTUPCB

Mit dem Assembler-Makro IEDTUPCB kann der Kontrollblock EDTUPCB generiert werden.

Name	Operation	Operanden
[name]	IEDTUPCB	[{ D / C }][,prefix] [,VERSION = { 2 / 3 }]

name – Symbolischer Name der 1. DS-Anweisung bei Angabe von C.
– Name der DSECT bei Angabe von D.

Wird *name* nicht angegeben, wird EDTUPCB benutzt (mit vorangestelltem *prefix*, falls angegeben).

D Es wird ein Pseudoabschnitt (DSECT) generiert.

C Es wird ein Speicherabschnitt mit symbolischen Adressen generiert (keine CSECT-Anweisung).

prefix 1 Zeichen, mit dem die generierten Feldnamen beginnen sollen.

Wird *prefix* nicht angegeben, wird standardmäßig E eingesetzt.

VERSION Auswahl, welche Version des Kontrollblocks generiert werden soll:

Die Version 2 wird mit dem V16-Format der Schnittstelle eingesetzt.
Die Version 3 wird mit dem V17-Format der Schnittstelle eingesetzt.

Bei Angabe des Makros `IEDTUPCB VERSION=3` wird der Kontrollblock `EDTUPCB` in folgender Form generiert:

```
          IEDTUPCB D,VERSION=3
1 EDTUPCB  MFPRE DNAME=EDT, MF=D
2 EDTUPCB  DS    0F
1 *--------------- EDT UNIT NUMBER, EDTUPCB VERSION NUMBER ---------------
1 EUPUNITC EQU   66                  EDT UNIT NUMBER
1 EUPVERSC EQU   3                   EDTUP VERSION NUMBER
1 EUPCMDM  EQU   (32763+4)           EDT COMMAND MAXLENGTH
1 EUPMSGM  EQU   (80+4)              EDT MESSAGE MAXLENGTH
1 *--------------- CONTROL BLOCK EDTUPCB --------------------------------
1 *                       *---- CONTROL BLOCK HEADER --------------------
1 EUPFHE   DS    0XL8                GENERAL OPERAND LIST HEADER
1 EUPIFID  DS    0A                  INTERFACE IDENTIFIER
1 EUPUNIT  DC    AL2(XUPUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EUPVERS  DC    AL1(XUPVERSC)       FUNCTION INTERFACE VERSION NUMBER
1          DS    A                   RESERVED
1 *                       *---- INHIBIT FLAGS ----------------------------
1 EUPINHBT DC    X'00'               INHIBIT FLAG BYTE
1 EUPMODE  EQU   X'80'               * NO SWITCH TO COMPATIBLE MODE V17.0
1 EUPNTXT  EQU   X'40'               * NO <TEXT>     (HALT / RETURN)
1 EUPN@EDO EQU   X'20'               * NO @EDIT ONLY (L-MODE : RDATA)
1 EUPN@EDT EQU   X'10'               * NO @EDIT      (L-MODE : WRTRD)
1 EUPNUSER EQU   X'08'               * NO USER-PROG. (@RUN/@USE)
1 EUPNBKPT EQU   X'04'               * NO BKPT (@SYSTEM)
1 EUPNCMDM EQU   X'02'               * NO CMD (@SYSTEM <STRING>)
1 EUPNEXEC EQU   X'01'               * NO MEXEC/MLOAD (@EXEC/@LOAD)
1 EUPNINHB EQU   X'00'               * NO RESTRICTIONS
1          DS    AL3                 RESERVED
1 *--------------- LENGTH OF CONTROL BLOCK ------------------------------
1 EUPUPCBL EQU   *-EDTUPCB
```

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EUPUNIT	Eindeutige Identifikation des EDT.	2	X	A(M)	
EUPVERS	Änderungsstand des Kontrollblocks.	1	X	A(M)	
EUPINHBT	Durch Setzen der einzelnen Bits kann man bestimmte Anweisungen für den Benutzer sperren, um ein undefiniertes Beenden des EDT bzw. das Verlassen des Unicode-Modus zu verhindern: EUPMODE Sperren des Umschaltens zwischen Unicode-Modus und Kompatibilitäts-Modus EUPNTXT Bei @HALT und @RETURN wird die Angabe von message gesperrt EUPN@EDO Sperren von @EDIT ONLY EUPN@EDT Sperren von @EDIT EUPNUSER Sperren von @USE und @RUN EUPNBKPT Sperren der @SYSTEM-Anweisung (ohne Operanden) EUPNCMDM Sperren der @SYSTEM-Anweisung (mit Operanden) EUPNEXEC Sperren der @EXEC / @LOAD-Anweisung	1		A	

A	Aufrufparameter	Muss vom Aufrufer versorgt werden.
(M)		Wird vom Makro (bei Angabe des Parameters C) gesetzt und sollte vom Anwender nicht verändert werden.
R	Rückkehrparameter	Wird vom EDT versorgt.
X	Binärformat	Binäre Zahlen.

Änderungen gegenüber dem V16-Format

Das Flag EUPMODE ist neu.

Kompatibles V17-Format

– Das Flag EUPMODE darf nicht gesetzt sein.

– Die Konstante EUPMSGM wurde nur aus Kompatibilitätsgründen beibehalten.
Die maximale Länge der Meldung wird dynamisch in Abhängigkeit vom Terminaltyp festgelegt.

3.2.3 EDTAMCB - Access-Method-Kontrollblock

Der EDTAMCB (*Access-Method-Control-Block*) ist der Kontrollblock für die logischen Satzzugriffsfunktionen. Er enthält jene Datenfelder, die bei einem Zugriff auf die Arbeitsdateien benötigt werden.

Erstellen des Kontrollblocks EDTAMCB

Mit dem Assembler-Makro IEDTAMCB kann der Kontrollblock EDTAMCB generiert werden.

Name	Operation	Operanden
[name]	IEDTAMCB	[{ D / C }][,prefix] [,VERSION = { 1 / 2 }]

name – Symbolischer Name der 1. DS-Anweisung bei Angabe von C.
– Name der DSECT bei Angabe von D.

Wird *name* nicht angegeben, wird EDTAMCB benutzt (mit vorangestelltem *prefix*, falls angegeben).

D Es wird ein Pseudoabschnitt (DSECT) generiert.

C Es wird ein Speicherabschnitt mit symbolischen Adressen generiert (keine CSECT-Anweisung).

prefix 1 Zeichen, mit dem die generierten Feldnamen beginnen sollen.

Wird *prefix* nicht angegeben, wird standardmäßig E eingesetzt.

VERSION Auswahl, welche Version des Kontrollblocks generiert werden soll:

Die Version 1 wird mit dem V16-Format der Schnittstelle eingesetzt.
Die Version 2 wird mit dem V17-Format der Schnittstelle eingesetzt.

Bei Angabe des Makros `IEDTAMCB VERSION=2` wird der Kontrollblock `EDTAMCB` in folgender Form generiert:

```
          IEDTAMCB D,VERSION=2
1 EDTAMCB  MFPRE DNAME=EDT,MF=D
2 EDTAMCB  DSECT ,
2               *,##### PREFIX=I, MACID= #####
1 *--------------- EDT UNIT NUMBER, EDTAMCB VERSION NUMBER --------------
1 EAMUNITC EQU   66                EDT UNIT NUMBER
1 EAMVERSC EQU   2                 INTERFACE IDENTIFIER
1 *--------------- CONTROL BLOCK EDTAMCB --------------------------------
1 *                     *--- CONTROL BLOCK HEADER ----------------------
1 EAMFHE   DS    0XL8              GENERAL OPERAND LIST HEADER
1 EAMIFID  DS    0A                INTERFACE IDENTIFIER
1 EAMUNIT  DC    AL2(XAMUNITC)     UNIT NUMBER
1          DS    AL1               RESERVED
1 EAMVERS  DC    AL1(XAMVERSC)     FUNCTION INTERFACE VERSION NUMBER
1          DS    A                 RESERVED
1          DS    X                 RESERVED                          V17.0
1 *                     *---- FLAG-BYTE --------------------------- V16.4
1 EAMFLAG  DC    X'00'             FLAG                              V16.4
1 EAMIGN13 EQU   X'01'             IGNORE LINE MARK 13               V16.4
1 EAMNOMOD EQU   X'02'             INHIBIT SETTING MODIFIED FLAG     V16.6
1          DS    AL2               RESERVED
1 *                     *--- INPUT PARAMETERS -------------------------
1 EAMFILE  DC    CL8' '            FILENAME
1 EAMDISP  DC    F'0'              DISPLACEMENT
1 EAMLKEY1 DC    H'8'              LENGTH OF KEY1
1 EAMLKEY2 DC    H'8'              LENGTH OF KEY2
1 *                     *--- INPUT PARAMETERS (ONLY IN MOVE MODE) ------
1 EAMPKEY  DC    H'8'              LENGTH OF KEY OUTPUT BUFFER
1 EAMPREC  DC    H'0'              LENGTH OF RECORD OUTPUTBUFFER
1 *                     *--- INPUT / OUTPUT PARAMETERS -----------------
1 EAMLKEY  DC    H'8'              LENGTH OF KEY
1 EAMLREC  DC    H'0'              LENGTH OF RECORD
1 EAMMARK  DS    0H                LINE MARKS (16 BITS)
1 EAMMARK2 DC    X'00'             UPPER MARKS (8 BITS)
1 EAMMK15  EQU   X'80'             MARK 15 (BIT 2**15)
1 EAMMK14  EQU   X'40'             MARK 14 (BIT 2**14)
1 EAMMK13  EQU   X'20'             MARK 13 (BIT 2**13)
1 EAMMK09  EQU   X'02'             MARK 09 (BIT 2**09)
1 EAMMK08  EQU   X'01'             MARK 08 (BIT 2**08)
1 EAMMARK1 DC    X'00'             LOWER MARKS (8 BIT)
1 EAMMK07  EQU   X'80'             MARK 07 (BIT 2**07)
1 EAMMK06  EQU   X'40'             MARK 06 (BIT 2**06)
1 EAMMK05  EQU   X'20'             MARK 05 (BIT 2**05)
1 EAMMK04  EQU   X'10'             MARK 04 (BIT 2**04)
```

```
1 EAMMK03  EQU   X'08'              MARK 03 (BIT 2**03)
1 EAMMK02  EQU   X'04'              MARK 02 (BIT 2**02)
1 EAMMK01  EQU   X'02'              MARK 01 (BIT 2**01)
1          DS    AL2                RESERVED
1 *--------------- LENGTH OF CONTROL BLOCK -------------------------------
1 EAMAMCBL EQU   *-EDTAMCB
```

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EAMUNIT	Eindeutige Identifikation des EDT.	2	X	A(M)	
EAMVERS	Änderungsstand des Kontrollblocks.	1	X	A(M)	
EAMFLAG	Das Byte enthält Flags zur Satzbearbeitung	1		A	
Flag EAMIGN13	Dieses Flag muß gesetzt sein, falls Sätze mit Markierung 13 gelesen werden sollen (IEDTGET) oder markiert werden sollen (IEDTPTM).			A	
Flag EAMNOMOD	Dieses Flag muß gesetzt sein, falls beim Schreiben eines Satzes die Arbeitsdatei nicht als modifiziert gekennzeichnet werden soll (IEDTPUT).			A	
EAMFILE	Angabe der Arbeitsdateivariablen ($0..$22), die angibt, auf welche Arbeitsdatei (0–22) zugegriffen werden soll oder der Werte G oder L0 bis L22 bei der Abfrage des Arbeitsdateistatus oder des Werts C beim Löschen des Kopierpuffers.	8	C	A	
EAMDISP	Dieses Feld enthält: – beim Lesen eines Satzes (IEDTGET) die relative Position des gewünschten Satzes – beim Lesen eines Satzes mit Markierung (IEDTGTM) die gewünschte Suchrichtung.	4	X	A	
EAMLKEY1	Länge des Puffers EDTKEY1	2	X	A	
EAMLKEY2	Länge des Puffers EDTKEY2	2	X	A	

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EAMPKEY	Dieses Feld muss vor dem Lesen eines Satzes (IEDTGET, IEDTGTM) mit der Länge des Ausgabepuffers für die Zeilennummer versorgt werden.Derzeit ist nur der Wert 8 zulässig.	2	X	A	
EAMPREC	Dieses Feld muss vor dem Lesen eines Satzes (IEDTGET, IEDTGTM) mit der Länge des Ausgabepuffers für den Satz versorgt werden.	2	X	A	
EAMLKEY	Länge der übertragenen Daten im Puffer EDTKEY. Beim Lesen übergibt der EDT die tatsächliche Länge der gelesenen Satznummer, beim Schreiben übergibt der Anwender die Länge der zu schreibenden Satznummer. Derzeit ist nur der Wert 8 zulässig.	2	X	A	R
EAMLREC	Länge der übertragenen Daten im Puffer EDTREC. Beim Lesen übergibt der EDT die tatsächliche Länge des gelesenen Satzes, beim Schreiben übergibt der Anwender die Länge des zu schreibenden Satzes.	2	X	A	R
EAMMARK	Angabe der Markierungen eines Satzes. Es wird sowohl bei der Eingabe als auch bei der Ausgabe verwendet. Das Markierungsfeld wird als Zusatzinformation zu jedem Satz verwaltet. Dem Benutzer stehen die Markierungen 1..9 (EAMMK01..EAMMK09) sowie die Markierungen 13, 14 und 15 EAMMK13, EAMMK14 und EAMMK15) bei IEDTPUT und IEDTPTM frei zur Verfügung, die restlichen Markierungen sind für Sonderfunktionen reserviert.	2		A	R

A	Aufrufparameter	Muss vom Aufrufer versorgt werden.
(M)		Wird vom Makro (bei Angabe des Parameters C) gesetzt und sollte vom Anwender nicht verändert werden.
R	Rückkehrparameter	Wird von EDT versorgt.
X	Binärformat	Binäre Zahlen.
C	Abdruckbar	Abdruckbare Texte (im Zeichensatz EDF03IRV).

Änderungen gegenüber dem V16-Format

– Das Feld EAMMMODB und die beiden Flags EAMMOVM und EAMLOCM entfallen.

– Die Equates für die nicht verwendeten Markierungen (EAMMK10, EAMMK11, EAMMK12 und EAMMK0) entfallen.

Kompatibles V17-Format

Keine Einschränkungen.

3.2.4 EDTPARG - Globale Einstellungen

Nach dem Lesen der globalen Einstellungen mit der Funktion IEDTGET legt der EDT die Informationen im Kontrollblock EDTPARG ab.

Erstellen des Kontrollblocks EDTPARG

Mit dem Assembler-Makro IEDTPARG kann der Kontrollblock EDTPARG generiert werden.

Name	Operation	Operanden
[name]	IEDTPARG	[{ D / C }][,prefix] [,VERSION = { 1 / 2 }]

name – Symbolischer Name der 1. DS-Anweisung bei Angabe von C.
– Name der DSECT bei Angabe von D.

Wird *name* nicht angegeben, wird EDTPARG benutzt (mit vorangestelltem *prefix*, falls angegeben).

D Es wird ein Pseudoabschnitt (DSECT) generiert.

C Es wird ein Speicherabschnitt mit symbolischen Adressen generiert (keine CSECT-Anweisung).

prefix 1 Zeichen, mit dem die generierten Feldnamen beginnen sollen.

Wird *prefix* nicht angegeben, wird standardmäßig E eingesetzt.

VERSION Auswahl, welche Version des Kontrollblocks generiert werden soll:

Die Version 1 wird mit dem V16-Format der Schnittstelle eingesetzt.
Die Version 2 wird mit dem V17-Format der Schnittstelle eingesetzt.

Bei Angabe des Makros IEDTPARG VERSION=2 wird der Kontrollblock EDTPARG in folgender Form generiert:

```
          IEDTPARG D,VERSION=2
1 EDTPARG  MFPRE DNAME=EDT,MF=D
2 EDTPARG  DSECT ,
2               *,##### PREFIX=I, MACID= #####
1 *--------------- EDT UNIT NUMBER, EDTPARL VERSION NUMBER --------------
1 EPGUNITC EQU   66                  EDT UNIT NUMBER
1 EPGVERSC EQU   2                   EDTPARG VERSION NUMBER
1 *--------------- CONTROL BLOCK EDTPARL --------------------------------
1 *                        *---- CONTROL BLOCK HEADER --------------------
1 EPGFHE   DS    0XL8                GENERAL OPERAND LIST HEADER
```

```
1 EPGIFID  DS    0A                  INTERFACE IDENTIFIER
1 EPGUNIT  DC    AL2(XPGUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EPGVERS  DC    AL1(XPGVERSC)       FUNCTION INTERFACE VERSION NUMBER
1          DS    A                   RESERVED
1 *                      *---- OUTPUT FIELDS ----------------------------
1 EPGMODE  DS    CL1                 EDT-MODE (F/L/C)
1 EPG@SYM  DS    CL1                 EDT-STATEMENT-SYMBOL
1 EPGWDS1  DS    H                   SIZE OF WINDOW 1
1 EPGWDS2  DS    H                   SIZE OF WINDOW 2
1 EPGFILE1 DS    CL8                 WORKFILE IN WINDOW 1
1 EPGFILE2 DS    CL8                 WORKFILE IN WINDOW 2
1          DS    XL10                RESERVED
1 *--------------- TOTAL LENGTH OF CONTROL BLOCK -----------------------
1 EPGPARGL EQU   *-EDTPARG
```

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EPGUNIT	Eindeutige Identifikation des EDT.	2	X	A(M)	
EPGVERS	Änderungsstand des Kontrollblocks.	1	X	A(M)	
EPGMODE	aktueller Modus: F = F-Modus L = L-Modus C = Caller (dh. aus einer Anweisung heraus, die über die IEDTGLE-Schnittstelle abgearbeitet wird).	1	C		R
EPG@SYM	EDT-Anweisungssymbol	1	C		R
EPGWDS1	Fenstergröße 1 (2..24)	2			R
EPGWDS2	Fenstergröße 2 (0..22)	2			R
EPGFIL1	Arbeitsdatei im 1.Fenster ($0..$22), mit Leerzeichen aufgefüllt	8	C		R
EPGFIL2	Arbeitsdatei im 2.Fenster ($0..$22), mit Leerzeichen aufgefüllt	8	C		R

A	Aufrufparameter	Muss vom Aufrufer versorgt werden.
(M)		Wird vom Makro (bei Angabe des Parameters C) gesetzt und sollte vom Anwender nicht verändert werden.
R	Rückkehrparameter	Wird von EDT versorgt.
X	Binärformat	Binäre Zahlen.
C	Abdruckbar	Abdruckbare Texte (im Zeichensatz EDF03IRV).

Änderungen gegenüber dem V16-Format

Das Feld EPGCCSN ist entfallen (jetzt mit dem Namen EPLCCSN im Kontrollblock EDTPARL). Bei Aufruf der Schnittstelle IEDTGET(PARG) muss das Feld EAMPREC im AMCB nicht mehr versorgt werden.

Bei Aufruf des EDT V17.0A im Unicode-Modus mit einem EDTPARG Kontrollblock der Version 1 wird das Feld EPGCCSN nur dann mit einem Wert ungleich Leerzeichen versorgt, wenn für alle nicht leeren Arbeitsdateien der gleiche Zeichensatz eingestellt ist.

Kompatibles V17-Format

Keine Einschränkungen.

3.2.5 EDTPARL - Arbeitsdateispezifische Einstellungen

Beim Lesen der arbeitsdateispezifischen Einstellungen mit der Funktion `IEDTGET` legt der EDT die Informationen im Kontrollblock `EDTPARL` ab.

Erstellen des Kontrollblocks EDTPARL

Mit dem Assembler-Makro `IEDTPARL` kann der Kontrollblock `EDTPARL` generiert werden.

Name	Operation	Operanden
[name]	IEDTPARL	[{ D / C }][,prefix] [,VERSION = { 1 / 2 / 3 / 4 }]

name – Symbolischer Name der 1. `DS`-Anweisung bei Angabe von `C`.
– Name der `DSECT` bei Angabe von `D`.

Wird *name* nicht angegeben, wird `EDTPARL` benutzt (mit vorangestelltem *prefix*, falls angegeben).

D Es wird ein Pseudoabschnitt (`DSECT`) generiert.

C Es wird ein Speicherabschnitt mit symbolischen Adressen generiert (keine `CSECT`-Anweisung).

prefix 1 Zeichen, mit dem die generierten Feldnamen beginnen sollen.

Wird *prefix* nicht angegeben, wird standardmäßig `E` eingesetzt.

VERSION Auswahl, welche Version des Kontrollblocks generiert werden soll:

Die Versionen 1, 2 und 3 werden mit dem V16-Format der Schnittstelle eingesetzt.
Die Version 4 wird mit dem V17-Format der Schnittstelle eingesetzt.

Bei Angabe des Makros `IEDTPARL VERSION=4` wird der Kontrollblock `EDTPARL` in folgender Form generiert:

```
          IEDTPARL D,VERSION=4
1 EDTPARL  MFPRE DNAME=EDT,MF=D,PREFIX=*
2 EDTPARL  DSECT ,
2               *,##### PREFIX=, MACID= #####
1 *--------------- EDT UNIT NUMBER, EDTPARL VERSION NUMBER --------------
1 EPLUNITC EQU   66                  EDT UNIT NUMBER
1 EPLVERSC EQU   4                   EDTPARL VERSION NUMBER
1 *                       *---- CONTROL BLOCK HEADER --------------------
1 EPLFHE   DS    0XL8                GENERAL OPERAND LIST HEADER
1 EPLIFID  DS    0A                  INTERFACE IDENTIFIER
1 EPLUNIT  DC    AL2(XPLUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EPLVERS  DC    AL1(XPLVERSC)       FUNCTION INTERFACE VERSION NUMBER
1          DS    A                   RESERVED
1 *                       *---- OUTPUT FIELDS ---------------------------
1 EPLVPOS  DS    CL8                 FIRST LINE IN WINDOW
1 EPLHPOS  DS    H                   FIRST COLUMN IN WINDOW
1 EPLRLIM  DS    H                   MAX RECORD-LENGTH IN F-MODE
1 EPLINF   DS    CL1                 INF ON/OFF  (1/0)
1 EPLLOW   DS    CL1                 LOWER ON/OFF (1/0)
1 EPLHEX   DS    CL1                 HEX ON/OFF (1/0)
1 EPLEDL   DS    CL1                 EDIT LONG ON/OFF (1/0)
1 EPLSCALE DS    CL1                 SCALE ON/OFF (1/0)
1 EPLPROT  DS    CL1                 PROTECTION ON/OFF (1/0)
1 EPLSTRUC DS    CL1                 STRUCTURE SYMBOL IF EBCDIC
1 EPLOPEN  DS    CL1                 OPEN FLAG:   (I/P/R/S/X/0)
1 EPLEMPTY DS    CL1                 EMPTY FLAG
1 EPLMODIF DS    CL1                 MODIFIED FLAG
1 EPLSTDF  DS    CL54                STANDARD FILENAME
1 EPLSTDL  DS    CL54                STANDARD LIBRARY NAME
1 EPLSTDT  DS    CL8                 STANDARD PLAM TYPE
1          DS    CL4                 RESERVED
1 EPLVPOS1 DS    CL8                 FIRST LINE IN WINDOW 1
1 EPLHPOS1 DS    H                   FIRST COLUMN IN WINDOW 1
1 EPLVPOS2 DS    CL8                 FIRST LINE IN WINDOW 2
1 EPLHPOS2 DS    H                   FIRST COLUMN IN WINDOW 2
1 EPLINDX1 DS    CL1                 INDEX OFF/ON/FULL (0/1/2) WINDOW 1
1 EPLINDX2 DS    CL1                 INDEX OFF/ON/FULL (0/1/2) WINDOW 2
1 EPLOPENC DS    XL1024              COMMON AREA FOR FILE DESCRIPTION
1 EPLOPEND EQU   *                   END OF COMMON AREA
1          ORG   XPLOPENC            DESCRIPTION OF OPENED DATA FILE
1 EPLOPNFL DS    CL54                NAME OF OPENED FILE/PLAM LIBRARY
1 EPLOPNE  DS    CL64                NAME OF OPENED PLAM ELEMENT
1 EPLOPNV  DS    CL24                VERSION OF OPENED PLAM ELEMENT
```

```
1 EPLOPNT  DS    CL8                TYP OF OPENED PLAM ELEMENT
1          ORG   XPLOPENC           DESCRIPTION OF OPENED UFS FILE
1 EPLOPNX  DS    CL1024             NAME OF OPENED UFS FILE
1 *
1 EPLCCSN  DS    CL8                CODED CHARACTER SET NAME       V17.0
1 EPLCCSNG DS    XL1                CCS IS GLOBAL (0/1)            V17.0
1 EPLSSTRU DS    H                  STRUCTURE SYMBOL UTF16
1 *-------------- LENGTH OF CONTROL BLOCK ------------------------------
1 EPLPARLL EQU   *-EDTPARL
```

Das Feld EPLOPENC enthält abhängig vom Open-Flag EPLOPEN die Beschreibung der mit @OPEN oder @XOPEN eröffneten Datei oder des PLAM-Elementes.

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EPLUNIT	Identifikation von EDT.	2	X	A(M)	
EPLVERS	Änderungsstand des Kontrollblocks.	1	X	A(M)	
EPLVPOS	1. Zeile im Datenfenster (00000001..99999999), gleicher Wert wie EPLVPOS1	8	C		R
EPLHPOS	1. Spalte im Datenfenster (1..32768), gleicher Wert wie EPLHPOS1)	2	X		R
EPLRLIM	Max. Satzlänge im F-Modus (1..32768)	2	X		R
EPLINF	Information On/Off (1/0)	1	C		R
EPLLOW	Lower On/Off (1/0)	1	C		R
EPLHEX	Hex On/Off (1/0)	1	C		R
EPLEDL	Edit Long On/Off (1/0)	1	C		R
EPLSCALE	Scale On/Off (1/0)	1	C		R
EPLPROT	Protection On/Off (1/0)	1	C		R
EPLSTRUC	Stuktur-Symbol (falls EBCDIC-codiert, bei anderen Codierungen wird der Wert nur in EPLSSTRU abgelegt)	1	C		R

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EPLOPEN	OPEN-Anzeige = R: ISAM real P: PLAM I: ISAM virtuell S: SAM virtuell X: UFS/POSIX 0: Keine Datei geöffnet	1	C		R
EPLEMPTY	Datei leer Ja/Nein (1/0)	1	C		R
EPLMODIF	Datei verändert Ja/Nein (1/0)	1	C		R
EPLSTDF	Standard-Dateiname, der mit @FILE eingestellt wurde	54	C		R
EPLSTDL	Standard-Bibliotheksname, der mit @PAR LIBRARY=… eingestellt wurde	54	C		R
EPLSTDT	Standard-Typ, der mit @PAR ELEMENT-TYPE=… eingestellt wurde	8	C		R
EPLOPNFL	EPLOPEN = R: ISAM-Dateiname, real geöffnet P: PLAM-Bibliotheksname I: ISAM-Dateiname S: SAM-Dateiname, aufgefüllt mit Leerzeichen	54	C		R
EPLOPNE	EPLOPEN = P: PLAM-Elementname, aufgefüllt mit Leerzeichen	64	C		R
EPLOPNV	EPLOPEN = P: PLAM-Versionsnummer, aufgefüllt mit Leerzeichen	24	C		R
EPLOPNT	EPLOPEN = P: PLAM-Typ, aufgefüllt mit Leerzeichen	8	C		R
EPLOPNX	EPLOPEN = X: POSIX-Dateiname, aufgefüllt mit Leerzeichen, nicht mit Null terminiert	1024	C		R
EPLVPOS1	Erste Zeilennummer im Fenster 1	8	C		R
EPLHPOS1	Erste Spalte im Fenster 1	2	X		R
EPLVPOS2	Erste Zeilennummer im Fenster 2	8	C		R
EPLHPOS2	Erste Spalte im Fenster 2	2	X		R

Bedeutung der Kontrollblockfelder		Länge (Byte)	Format	Parameterart	
				Aufruf	Rückkehr
EPLINDX1	Index Off/On/Full Fenster 1 (0/1/2), wie mit @PAR INDEX=...bzw. @PAR EDIT-FULL=... eingestellt wurde	1	C		R
EPLINDX2	Index Off/On/Full Fenster 2 (0/1/2)	1	C		R
EPLCCSN	Zeichensatz der Arbeitsdatei. Ist die Arbeitsdatei leer, enthält EPLCCSN Leerzeichen.	8	C		R
EPLCCSNG	Der Zeichensatz gilt für alle nicht-leeren Arbeitsdatein Ja/Nein (1/0)	1	C		R
EPLSSTRU	Struktursymbol in UTF16	2	U		R

A	Aufrufparameter	Muss vom Aufrufer versorgt werden.
(M)		Wird vom Makro (bei Angabe des Parameters C) gesetzt und sollte vom Anwender nicht verändert werden.
L	Aufrufparameter	Muss vom Aufrufer beim ersten Aufruf mit binären Nullen gelöscht werden.
R	Rückkehrparameter	Wird von EDT versorgt.
X	Binärformat	Binäre Zahlen.
C	Abdruckbar	Abdruckbare Texte (im Zeichensatz EDF03IRV).
U	UTF16-Zeichen	Text in UTF16-Codierung

Änderungen gegenüber dem V16-Format

– Das Feld EPLCCSN ist neu (früher mit dem Namen EPGCCSN im Kontrollblock EDTPARG).

– Das Feld EPLCCSNG ist neu.

– Das Feld EPLSTCOD ist entfallen.

– Das Feld EPLOPNXC ist entfallen.

– Das Feld EPLSTRUC enthält nur noch dann das Struktur-Symbol, wenn dieses als EBCDIC-Zeichen darstellbar ist, sonst enthält es binäre Nullen. Das Feld EPLSSTRU enthält in jedem Fall das Struktur-Symbol in UTF16-Codierung.

Kompatibles V17-Format

Falls das V17-Format zum Aufruf des EDT im Kompatibilitäts-Modus oder eines EDT kleiner V17.0 benutzt wird, werden die von der jeweiligen V16-Version nicht gelieferten Informationen mit Standardwerten versorgt:

– Alle Dateinamensfelder mit Leerzeichen
– EPLVPOS1 und EPLVPOS2 mit '00010000'
– EPLHPOS1 und EPLHPOS2 mit binär 1
– EPLINDX1 und EPLINDX2 mit '1'
– EPLCCSN mit dem global eingestellten CCSN (EPLCCSNG enthält dann 1), falls die Arbeitsdatei nicht leer ist, andernfalls mit Leerzeichen (EPLCCSNG enthält dann 0).

3.3 Puffer

Die benutzten Namen der Puffer (EDTREC, EDTKEY, EDTKEY1, EDTKEY2, COMMAND) sind nur Platzhalter für die Beschreibung. Der Anwender kann in seinem Programm die Namen der Puffer frei wählen.

Werden bei einer Funktion mehrere Puffer benutzt, müssen sie im gleichen Zeichensatz codiert sein.

3.3.1 Satz (EDTREC)

Der Puffer wird verwendet

- zur Übergabe des Satzes durch den EDT beim Lesen eines Satzes (Funktionen IEDTGET, IEDTGTM) oder
- zur Übergabe des Satzes an den EDT beim Schreiben eines Satzes (Funktion IEDTPUT).

Die Länge eines Satzes beträgt minimal 0 Byte, maximal 32768 Byte und enthält *kein* Satzlängenfeld. Die Länge wird im Kontrollblock EDTAMCB spezifiziert.

3.3.2 Zeilennummer (EDTKEY, EDTKEY1, EDTKEY2)

Diese Puffer enthalten eine Zeilennummer.

Eine Zeilennummer ist 8 Byte lang. Der zulässige Bereich für Zeilennummern in einer Arbeitsdatei erstreckt sich

von C'00000001' (hexadezimal 'F0F0F0F0F0F0F0F1')

bis C'99999999' (hexadezimal 'F9F9F9F9F9F9F9F9')

Zum Ansprechen des ersten bzw. des letzten Satzes einer Arbeitsdatei kann bei den Zugriffsfunktionen IEDTGET, IEDTGTM und IEDTDEL der Eingabeparameter auch binär in folgender Form angegeben werden:

für den ersten Satz binäre Nullen (hexadezimal '0000000000000000')

für den letzten Satz binäre Einsen (hexadezimal 'FFFFFFFFFFFFFFFF')

Die Puffer EDTKEY1, EDTKEY2 und EDTKEY dürfen nur mit Werten aus dem angegebenen Bereich versorgt werden. Die zugehörigen Längenfelder im EDTAMCB müssen vom Benutzer mit dem Wert 8 versorgt werden. Angaben außerhalb dieses Bereichs liefern einen Fehlercode.

3.3.3 Anweisungsfolge in einem Puffer (COMMAND)

Bei den Funktionen IEDTCMD und IEDTEXE enthält der Puffer eine Anweisungsfolge, die vom EDT ausgeführt werden soll.

Beim Aufruf einer benutzerdefinierten Anweisung oder einer Anwenderroutine enthält der Puffer den Text, der beim Aufruf angegeben wurde.

Beim Aufruf eines Anweisungsfilters enthält der Puffer die zu filternde Anweisung.

Er hat folgendes Format: 2 Byte Länge von COMMAND (Gesamtlänge einschließlich der des Längen- und ungenutzten Feldes), 2 Byte ungenutzt, n Byte Satzinhalt.

Die Länge beträgt max. 32767 Bytes (4+32763).

3.3.4 Meldungen in einem Puffer (MESSAGE1, MESSAGE2)

Diese Puffer enthalten je eine Meldung, die bei der Funktion IEDTCMD ausgegeben werden soll.

Sie haben folgendes Format: 2 Byte Länge von MESSAGE (Gesamtlänge einschließlich der des Längen- und ungenutzten Feldes), 2 Byte ungenutzt, n Byte Satzinhalt.

Das Längenfeld muss die Länge in Bytes enthalten. Es werden jedoch max. 132 Zeichen bzw. 80 Zeichen (je nach aktueller Zeilenlänge des Bildschirms) dargestellt.

Dies wird durch das nur noch aus Kompatibilitätsgründen unterstützte *Equate* EUPMSGM nur ungenügend wiedergegeben.

3.4 Anweisungsfunktionen

Mit den Anweisungsfunktionen können:

– eine EDT-Version ausgewählt werden

– Informationen über die Versionsnummer des EDT abgefragt werden

– dem EDT eine Anweisung bzw. Anweisungsfolge übergeben werden.

Es stehen folgende Funktionen zur Verfügung:

Funktionen	Einsprungadresse
Lesen der Versionsnummer des EDT bzw. Auswahl der EDT-Version	IEDTINF
Ausführen von EDT-Anweisungen	IEDTCMD
Ausführen von EDT-Anweisungen ohne Bildschirmdialog	IEDTEXE

3.4.1 IEDTINF - Lesen der Versionsnummer des EDT

An der Schnittstelle IEDTINF erhält der Benutzer die Versionsnummer des geladenen EDT. Ist noch kein EDT geladen, wird er geladen aber noch nicht initialisiert. Ist der EDT unter IMON installiert, kann die gewünschte EDT-Version angegeben werden.

Die Version des nachzuladenden EDT wird im Feld EGLRMSGF angegeben (abdruckbar). Die Länge der Versionsangabe muß im Feld EGLRMSGL angegeben werden.

Format der Versionsnummer: EDT Vaa.a[d[ii]] wobei a und i Ziffern sind und d ein Buchstabe ist

Eine Versionsangabe wird aufgrund einer Längenangabe im Feld EGLRMSGL erkannt, wobei der Wert im erlaubten Bereich zwischen 9 und 12 liegen muß. Ohne Versionsangabe wird bei Koexistenz mehrerer Versionen die durch /SELECT-PRODUCT-VERSION eingestellte bzw. die höchste EDT-Version nachgeladen.

Kann die angegebene Version nicht nachgeladen werden, wird im Feld EGLMRET der Returncode EUPVEERR gesetzt, und der EDT wird nicht nachgeladen. Kann die STD-Version ermittelt werden, so wird diese im Feld EGLRMSGF eingetragen. Kann die STD-Version nicht festgestellt werden, wird im Feld EGLSR1 der Subreturncode EUPVE04 gesetzt.
Bei erfolgreichem Aufruf wird der Returncode EUPRETOK im Feld EGLMRET (EDTGLCB) gesetzt.

Der EDT übergibt im Feld EGLRMSGF die geladene EDT-Version im oben beschriebenen Format (unabhängig davon, ob beim Aufruf von der EDT geladen wurde oder schon vorher geladen war).

Wenn der EDT schon geladen ist, wird immer die Version des geladenen EDT zurückgeben, auch wenn der Aufrufer eine andere Version spezifiziert hat. Der Returncode ist auch in diesem Fall EUPRETOK.

Aufruf

Folgende Angaben sind notwendig:

– Versorgen der benötigten Felder im Kontrollblock EDTGLCB
– Aufruf der Einsprungadresse IEDTINF mit der Parameterliste

Übersichtstabelle

(Kontrollblock siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23).

Einsprungadresse	:	IEDTINF

Parameterliste	:	A(EDTGLCB)

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
	EGLINDB		EGLINFM
	EGLRMSG		

Hinweis

Im Unicode-Modus übergibt der EDT im Feld EGLINFM immer den Wert 0.

Returncodes

EGLMRET	EGLSR1
EUPRETOK EUPEDERR EUPPAERR EUPVEERR	EUPOK00 00 EUPPA04 EUPVE00 EUPVE04

Die Felder EGLMRET und EGLRS1 sind Felder des Kontrollblocks EDTGLCB. Bedeutung der Returncodes siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Der Kontrollblock EDTGLCB wird folgendermaßen deklariert und initialisiert:

```
iedglcb glcb = IEDGLCB_INIT;
```

Im C-Programm wird die Funktion IEDTINF folgendermaßen aufgerufen:

```
IEDTINF(&glcb);
```

3.4.2 IEDTCMD - Ausführen von EDT-Anweisungen

Bei diesem Aufruf wird dem EDT eine Anweisung bzw. Anweisungsfolge zur Ausführung übergeben.
Ist das Anweisungsfeld leer (Länge 4), wird sofort zum rufenden Programm zurückgekehrt.

Folgende Anweisungen sind an der IEDTCMD-Schnittstelle erlaubt:

@+	@GETVAR	@SEQUENCE
@-	@HALT	@SET
@:	@INPUT (Format 1, 2)	@SETF
@AUTOSAVE	@LIMITS	@SETJV
@BLOCK	@LIST	@SETLIST
@CHECK (Format 2)	@LOAD	@SETSW
@CLOSE	@LOG	@SETVAR
@CODENAME	@LOWER	@SHOW
@COLUMN	@MODE	@SORT
@COMPARE	@MOVE	@STAJV
@CONVERT	@ON	@STATUS
@COPY	@OPEN	@SUFFIX
@CREATE (Format 1+ 2)	@P-KEYS	@SYMBOLS
@DELETE	@PAGE	@SYNTAX
@DELIMIT	@PAR	@SYSTEM
@DIALOG	@PREFIX	@TABS
@DO (Format 1)	@PRINT	@TMODE
@DROP	@QUOTE	@UNLOAD
@EDIT (Format 1, 2, 3)	@RANGE	@UNSAVE
@ELIM	@READ	@USE
@ERAJV	@RENUMBER	@VDT
@EXEC	@RESET	@VTCSET
@FILE	@RETURN	@WRITE
@FSTAT	@RUN	@XCOPY
@GET	@SAVE	@XOPEN
@GETJV	@SEARCH-OPTION	@XWRITE
@GETLIST	@SEPARATE	

Außerdem sind die benutzerdefinierten Anweisungen erlaubt (siehe Kapitel „Benutzerdefinierte Anweisungen - @USE“ auf Seite 91).

Die Anweisung @EDIT (mit Ausnahme des Formats 4 - @EDIT LONG...) wird an der `IEDTCMD`-Schnittstelle grundsätzlich als @EDIT ONLY interpretiert und bewirkt den Übergang in den Line-Modus-Dialog.

Das EDT-Anweisungssymbol muß nicht angegeben werden (außer bei `@:`).

Der Programmlauf (Initialisierung, Übergang zum Benutzerdialog, Beendigung mit Entladen und Speicherfreigabe) wird durch die Anweisungsfolge gesteuert, die an den EDT übergeben wird.

Nach dem Laden von EDT wird dessen Datenbereich initialisiert (nur beim 1.Aufruf).

Nach dem Ausführen der Anweisungsfolge kehrt der EDT zum rufenden Programm zurück.

@HALT in der übergebenen Anweisungsfolge bewirkt das Beenden des EDT (Speicherfreigabe und Entladen).

Ist die Anweisungsfolge nicht mit @HALT abgeschlossen, wird ohne Freigabe des Datenbereichs zum rufenden Programm zurückgekehrt. Durch einen erneuten Aufruf mit einer Anweisungsfolge kann die Verarbeitung fortgesetzt oder durch Übergabe von @HALT der EDT beendet werden.

Innerhalb einer Routine, die eine benutzerdefinierte Anweisung bearbeitet, bzw. innerhalb einer Anwenderroutine (siehe Kapitel „Benutzerdefinierte Anweisungen - @USE“ auf Seite 91 und „Anwenderroutinen - @RUN“ auf Seite 101) ist ein Aufruf der IEDTCMD-Schnittstelle nur dann zulässig, wenn er benutzt wird, eine andere als die aufrufende Instanz des EDT anzusprechen.

Anweisungen an die aufrufende EDT-Instanz, bei denen der an die Anweisungsroutine übergebene globale Kontrollblock EDTGLCB verwendet wird, dürfen nur über die IEDTEXE-Schnittstelle erfolgen. Ein Aufruf der IEDTCMD-Schnittstelle aus einer Anweisungsroutine mit dem EDTGLCB der aufrufenden Instanz wird mit dem Returncode EUPPAERR/EUPPA08 abgewiesen. Weitere Informationen dazu finden sich im Kapitel „Benutzerdefinierte Anweisungen - @USE“ auf Seite 91 und den folgenden.

Tritt ein Fehler auf (Syntax- oder Laufzeitfehler), wird die Abarbeitung sofort mit einem entsprechenden Returncode und einer Fehlermeldung unterbrochen. In diesen Fällen dient das Feld EGLCMDS (in EDTGLCB) als Fehlerzeiger. Dieser zeigt auf den Beginn der fehlerhaften Anweisung innerhalb der Anweisungsfolge. Aus Kompatibilitätsgründen wird das erste Zeichen nach dem Satzlängenfeld mit '1' nummeriert (das erste Zeichen der übergebenen Anweisungsfolge hat bei dieser Zählung die Nummer '3'). Die Zählung erfolgt grundsätzlich in Zeichen, nicht in Bytes. Es wird der Returncode EUPSYERR bzw. EUPRTERR übergeben.

Benutzerdialog

Der Wechsel zum Benutzerdialog kann durch die Anweisung @DIALOG (Bildschirmdialog) bzw. durch @EDIT ONLY (Line-Modus-Dialog) innerhalb der übergebenen Anweisungsfolge erfolgen.

Jedes Mal, wenn mit der Anweisung @DIALOG aus der übergebenen Anweisungsfolge heraus in den Bildschirmdialog gewechselt wird, werden die übergebenen Meldungen (MESSAGE1, MESSAGE2) in den Meldungszeilen angezeigt.

Mit @EDIT ONLY wird in den Line-Modus-Dialog (Lesen mit RDATA) umgeschaltet.

Der Benutzerdialog wird mit @END, @HALT oder @RETURN beendet, im F-Modus auch mit K1.

Der EDT übergibt einen Returncode an den globalen Kontrollblock EDTGLCB (EGLRETC). Nach Beenden des Benutzerdialogs wird die Abarbeitung der Anweisungsfolge fortgesetzt. Die Anweisung @END setzt den gleichen Returncode wie @HALT.

Bei @HALT und @RETURN mit der Angabe von <message> wird der Text zusätzlich im Meldungsfeld EGLRMSGF im Kontrollblock EDTGLCB hinterlegt.

Wenn der Dialog mit @HALT ABNORMAL beendet wurde, wird der Mainreturncode EUPABERR gesetzt.

Ist das Flag EUPNTXT im EDTUPCB gesetzt, wird die Angabe von message bzw. ABNORMAL mit Fehlermeldung (im Dialog) zurückgewiesen.

Das Flag EGLSTXIT im EDTGLCB wird bei jedem Aufruf über die IEDTCMD-Schnittstelle ausgewertet. Bei Rückkehr zum aufrufenden Programm werden die Unterbrechungsroutinen des EDT - falls sie angefordert wurden - wieder abgemeldet.

Durch Setzen des Flags EUPNUSER im EDTUPCB wird die Verarbeitung einer @USE-Anweisung im Dialog abgewiesen.

Kontrollstrukturen

Folgende Datenbereiche müssen vor dem Aufruf der Funktion definiert werden:

- der Kontrollblock EDTGLCB
- der Kontrollblock EDTUPCB
- die Anweisungsfolge (COMMAND)
- optional 2 Meldungszeilen (MESSAGE1 und MESSAGE2), sonst Nullzeiger

Die Kontrollblöcke sind im Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22. Die Puffer COMMAND, MESSAGE1 und MESSAGE2 sind im Abschnitt Puffer beschrieben.

Bei ungeteiltem Bildschirm wird im Dialog MESSAGE1 in der Meldungszeile angezeigt- bei geteiltem Bildschirm wird MESSAGE1 in der ersten, MESSAGE2 in der zweiten Meldungszeile angezeigt.

Enthält das Längenfeld von MESSAGE1 oder MESSAGE2 einen Wert, der kleiner oder gleich 4 ist, wird die Ausgabe der entsprechenden Meldungszeile unterdrückt. Beim erstmaligen Übergang in den benutzerdialog wird in diesem Fall die EDT-Startmeldung ausgegeben. Dies gilt sinngemäß auch bei Angabe eines Nullzeigers.

Wenn @DIALOG bzw. @EDIT ONLY nicht als letzte Anweisung in der Anweisungsfolge steht, ist zu beachten, dass Returncode und Meldung im EDTGLCB von nachfolgenden Anweisungen verursacht werden können.

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

- Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTUPCB.
- Versorgen des Puffers COMMAND mit der Anweisungsfolge.
- Versorgen der Puffer MESSAGE1 und MESSAGE2 mit Meldungstexten bzw. der entsprechenden Felder der Parameterliste mit Nullzeigern.
- Aufruf der Einsprungadresse IEDTCMD mit der Parameterliste.

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23).

Einsprungadresse	:	IEDTCMD

Parameterliste	:	A (EDTGLCB, EDTUPCB, COMMAND, MESSAGE1, MESSAGE2)

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
	EGLINDB		EGLCMDS
	EGLCCSN		EGLFILE
EDTUPCB:	EUPUNIT		
	EUPVERS		
	EUPINHBT		
COMMAND			
MESSAGE1 / NULL			
MESSAGE2 / NULL			

Hinweis

Bei jeder Rückkehr werden im Kontrollblock EDTGLCB der Returncode und der Name der aktuellen Arbeitsdatei (EGLFILE) eingetragen.

Returncodes

EGLMRET	EGLRS1
EUPRETOK	EUPOK00
	EUPOK04
	EUPOK08
	EUPOK12
	EUPOK16
	EUPOK20
EUPSYERR	00
EUPRTERR	00
EUPEDERR	00
EUPOSERR	00
EUPUSERR	00
EUPPAERR	EUPPA04
	EUPPA08
	EUPPA12
	EUPPA16
	EUPPA20
	EUPPA24
EUPSPERR	00
EUPABERR	EUPOK08

EGLMRET und EGLRS1 sind Felder des Kontrollblocks EDTGLCB. Die Bedeutung der Returncodes ist im Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23 beschrieben.

Beispiel

```
*****************************************************************
* CMDBSP:  BEISPIEL FUER AUSFUEHREN EINER EDT-ANWEISUNGSFOLGE   *
*          IM UNICODE-MODUS                                     *
*          (PAR SPLIT=OFF,LOWER=ON,SCALE=ON,INDEX=ON;DIALOG)    *
*****************************************************************
*
CMDBSP   START
CMDBSP   AMODE ANY
CMDBSP   RMODE ANY
         BALR  R10,0
         USING *,R10
         MVC   EGLCCSN,CCSN041
         LA    R13,SAVEAREA
         LA    R1,CMDPL
         L     R15,=V(IEDTCMD)
```

```
         BALR  R14,R15
         TERM  ,
*
* DATENBEREICH
R1       EQU   1
R10      EQU   10
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
SAVEAREA DS    18F
* - KONTROLLBLOECKE (EDTGLCB, EDTUPCB)
         IEDTGLCB C,VERSION=2
         IEDTUPCB C,VERSION=3
* - ANWEISUNGSFOLGE (COMMAND)
CMDDIA   DC    Y(CMDDIAL)
         DC    CL2' '
         DC    C'PAR SPLIT=OFF,LOWER=ON,SCALE=ON,INDEX=ON;DIALOG'
CMDDIAL  EQU   *-CMDDIA
* - MELDUNGSZEILE (MESSAGE1)
MSG1DIA  DC    Y(MSG1DIAL)
         DC    CL2' '
         DC    C'DIALOGENDE MIT HALT ODER <K1>'
MSG1DIAL EQU   *-MSG1DIA
* - MELDUNGSZEILE (MESSAGE2)
MSG2DIA  DC    Y(MSG2DIAL)
         DC    CL2' '
MSG2DIAL EQU   *-MSG2DIA
* - PARAMETERLISTE FUER CMD
CMDPL    DC    A(EDTGLCB)
         DC    A(EDTUPCB)
         DC    A(CMDDIA)
         DC    A(MSG1DIA)
         DC    A(MSG2DIA)
*
CCSN041  DC    CL8'EDF041 '
*
         END   CMDBSP
```

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Die Kontrollblöcke `EDTGLCB` und `EDTUPCB` werden folgendermaßen deklariert und initialisiert:

```
iedglcb glcb = IEDGLCB_INIT;
iedupcb upcb = IEDUPCB_INIT;
```

Für den Aufbau und die Versorgung der Strukturen `command`, `message1` und `message2` findet sich ein Beispiel im Abschnitt „Beispiel 1 - C-Hauptprogramm“ auf Seite 111.

Aufgerufen wird die Funktion `IEDTCMD` mit den Adressen dieser Strukturen:

```
IEDTCMD(&glcb,&upcb,&command,&message1,&message2);
```

3.4.3 IEDTEXE - Ausführen von EDT-Anweisungen ohne Bildschirmdialog

Bei diesem Aufruf wird dem EDT eine Anweisung bzw. Anweisungsfolge zur Ausführung übergeben.

Grundsätzliche Unterschiede zur IEDTCMD-Funktion:

– Der EDT muß bereits geladen und initialisiert sein.
– Es kann kein Bildschirmdialog geführt werden (@DIALOG und @EDIT sind nicht erlaubt).
– Der EDT kann nicht beendet bzw. entladen werden (@HALT, @RETURN und @MODE sind nicht erlaubt).
– Es kann keine EDT-Prozedur gestartet werden (@INPUT und @DO sind nicht erlaubt).

Folgende Anweisungen sind an der IEDTEXE-Schnittstelle erlaubt:

@+	@LIST	@SETJV
@-	@LOAD	@SETLIST
@BLOCK	@LOG	@SETSW
@CHECK (Format 2)	@LOWER	@SETVAR
@CLOSE	@MOVE	@SHOW
@CODENAME	@ON	@SORT
@COLUMN	@OPEN	@STAJV
@COMPARE	@P-KEYS	@STATUS
@CONVERT	@PAGE	@SUFFIX
@COPY	@PAR	@SYMBOLS
@CREATE (Format 1+ 2)	@PREFIX	@SYSTEM
@DELETE	@PRINT	@TABS
@DELIMIT	@QUOTE	@TMODE
@ELIM	@RANGE	@UNLOAD
@ERAJV	@READ	@UNSAVE
@EXEC	@RENUMBER	@USE
@FILE	@RESET	@VDT
@FSTAT	@SAVE	@VTCSET
@GET	@SEARCH-OPTION	@WRITE
@GETJV	@SEPARATE	@XCOPY
@GETLIST	@SEQUENCE	@XOPEN
@GETVAR	@SET	@XWRITE
@LIMITS	@SETF	

Im Gegensatz zum Verhalten des EDT V16.6 wird das Flag EGLSTXIT im EDTGLCB auch bei jedem Aufruf über die IEDTEXE-Schnittstelle ausgewertet. Bei Rückkehr zum aufrufenden Programm werden die Unterbrechungsroutinen des EDT, falls sie angefordert wurden, wieder abgemeldet. Ist das aufrufende Programm eine Anweisungs- oder Anwenderroutine, wird bei Rückkehr zum EDT bezüglich der Unterbrechungsbehandlung der Zustand wiederhergestellt, wie er vor dem Aufruf der externen Routine war.

Tritt ein Syntax- oder Laufzeitfehler auf, wird die Abarbeitung sofort mit einem entsprechenden Returncode und einer Fehlermeldung unterbrochen. Bei einem Syntaxfehler dient das Feld EGLCMDS (EDTGLCB) als Fehlerzeiger. Dieser zeigt auf den Beginn der fehlerhaften Anweisung innerhalb der Anweisungsfolge. Aus Kompatibilitätsgründen wird das erste Zeichen nach dem Satzlängenfeld mit '1' nummeriert (das erste Zeichen der übergebenen Anweisungsfolge hat bei dieser Zählung die Nummer '3'). Die Zählung erfolgt grundsätzlich in Zeichen, nicht in Bytes. Es wird der Returncode EUPSYERR übergeben oder EUPRTERR übergeben.

Im Gegensatz zur IEDTCMD-Schnittstelle darf die IEDTEXE-Schnittstelle auch innerhalb einer Routine, die eine benutzerdefinierte Anweisung bearbeitet, bzw. innerhalb einer Anwenderroutine (siehe Abschnitte Benutzerdefinierte Anweisungen - @USE und Anwenderroutinen - @RUN) für Anweisungen an die aufrufende Instanz des EDT verwendet werden.

Beim Aufruf der IEDTEXE-Funktion aus der Anweisungsroutine einer benutzerdefinierten Anweisung darf keine weitere benutzerdefinierte Anweisung eingegeben werden.

Kontrollstrukturen

Folgende Datenbereiche müssen vor dem Aufruf der IEDTEXE-Funktion in der Benutzerroutine definiert werden:

– der Kontrollblock (EDTGLCB)
– die Anweisung oder Anweisungsfolge (COMMAND)

Der Kontrollblock EDTGLCB ist im Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23 beschrieben. Der Puffer COMMAND ist im Abschnitt „Anweisungsfolge in einem Puffer (COMMAND)“ auf Seite 51 beschrieben.
In der Anweisungsroutine einer benutzerdefinierten Anweisung sollte der Kontrollblock EDTGLCB verwendet werden, der vom EDT übergeben wurde.

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

– Versorgen der Kontrollblockfelder im EDTGLCB
– Versorgen des Datenfeldes COMMAND mit der Anweisungsfolge
– Aufruf der Einsprungadresse IEDTEXE mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Einsprungadresse	:	IEDTEXE

Parameterliste	:	A (EDTGLCB, COMMAND)

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
	EGLINDB		EGLCMDS
	EGLCCSN		EGLFILE
			EGLUSR1
			EGLUSR2
COMMAND			EGLUSR3

Returncodes

EGLMRET	EGLRS1
EUPRETOK	00
EUPSYERR	00
EUPRTERR	00
EUPEDERR	00
EUPOSERR	00
EUPUSERR	00
EUPPAERR	EUPPA04
	EUPPA12
	EUPPA20

EGLMRET und EGLRS1 sind Felder des Kontrollblocks EDTGLCB.
Bedeutung der Returncodes siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Auch im C-Programm wird die Funktion `IEDTEXE` mit der Adresse von `EDTGLCB` und dem auszuführenden Kommando aufgerufen:

```
IEDTEXE(&glcb,&command);
```

3.5 Logische Satzzugriffsfunktionen

Mit den logischen Satzzugriffsfunktionen greift der Benutzer aus einem Benutzerprogramm auf die Sätze der Arbeitsdateien zu.

Es stehen folgende Zugriffsfunktionen zur Verfügung:

Zugriffsfunktionen	**Einsprungadresse**
Lesen eines Satzes	IEDTGET
Lesen eines markierten Satzes	IEDTGTM
Schreiben eines Satzes	IEDTPUT
Markieren eines Satzes	IEDTPTM
Löschen des Kopierpuffers oder eines Satzbereichs	IEDTDEL
Ändern einer Zeilennummer	IEDTREN
Lesen der globalen Einstellungen des EDT	IEDTGET
Lesen der arbeitsdateispezifischen Einstellungen des EDT	IEDTGET

Die einzelnen Zugriffsfunktionen sind in den folgenden Abschnitten beschrieben.

Eine Zugriffsfunktion kann nur ausgeführt werden, wenn der EDT initialisiert ist. Die Initialisierung erfolgt durch einen Aufruf der Funktion IEDTCMD (siehe Abschnitt „IEDTCMD - Ausführen von EDT-Anweisungen“ auf Seite 55).

Erfolgt der Aufruf in einer Anweisungs- oder Anwenderroutine mit dem mitgelieferten EDTGLCB, ist der EDT bereits initialisiert.

Werden Satzzugriffsfunktionen auf eine gerade aktive Arbeitsdatei angewandt, die mit @DO als Prozedur abgearbeitet wird, werden sie mit dem Returncode EAMACERR/EAMAC48 (Arbeitsdatei ist aktiv) abgewiesen.

Bearbeiten von Dateien und Bibliothekselementen

Die logischen Satzzugriffsfunktionen beziehen sich immer auf Sätze einer Arbeitsdatei. Sollen Dateien bzw. Bibliothekselemente bearbeitet werden, müssen diese ggf. vorher in eine Arbeitsdatei eingelesen werden (z.B. mit der Funktion IEDTCMD). Eine Datei kann auch real geöffnet sein. Das Zurückschreiben dieser Dateien bzw. Elemente wird mit EDT-Anweisungen vorgenommen (z.B. mit der Funktion IEDTCMD).

Jeder Satz hat eine Zeilennummer, über die auf ihn zugegriffen werden kann.
Jeder Satz kann auch verschiedene Markierungen haben (siehe [1], Abschnitt Satzmarkierungen).

Kontrollstrukturen

Vor dem Aufruf einer Satzzugriffsfunktion müssen die benötigten Kontrollblöcke im rufenden Programm deklariert und ggf. definiert und versorgt werden.

Returncodes der Satzzugriffsfunktionen

Die Tabelle gibt an, welche Returncodes der EDT bei den einzelnen Zugriffsfunktionen setzen kann, dabei steht `GET` für `IEDTGET`, usw.

Returncode		**Funktion**					
EGLMRET	EGLRS1	GET	GTM	PUT	PTM	REN	DEL
EAMRETOK	EAMOK00	x	x	x	x	x	x
	EAMOK04	x	x				
	EAMOK08	x	x				
	EAMOK12	x	x				
	EAMOK16						x
	EAMOK20						x
EAMACERR	EAMAC04			x			
	EAMAC08	x	x				
	EAMAC12	x	x				
	EAMAC16	x	x		x	x	x
	EAMAC20		x			x	
	EAMAC28		x	x	x		
	EAMAC32				x		
	EAMAC36					x	
	EAMAC40					x	
	EAMAC44					x	
	EAMAC48	x	x	x	x	x	x
EAMEDERR		x	x	x	x	x	x
EAMOSERR		x	x	x	x	x	x
EAMUSERR		x	x	x	x	x	x
EAMPAERR	EAMPA04	x	x	x	x	x	x
	EAMPA08	x	x	x	x	x	x
	EAMPA12	x	x	x	x	x	x
	EAMPA20	x	x	x	x	x	x
	EAMPA24	x	x	x	x	x	x
	EAMPA28			x			
	EAMPA32	x	x	x	x	x	x
	EAMPA36	x	x	x	x	x	x
	EAMPA40	x	x	x			
	EAMPA44	x	x	x			

Die Felder EGLMRET und EGLRS1 sind Felder des Kontrollblocks EDTGLCB. Bedeutung der Returncodes siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

3.5.1 IEDTGET - Lesen eines Satzes

Mit dieser Zugriffsfunktion kann ein Satz aus einer Arbeitsdatei gelesen werden.

Der zu lesende Satz wird durch folgende Angaben bestimmt:

– Arbeitsdatei ($0..$22) im Feld EAMFILE (EDTAMCB): Arbeitsdatei aus der der Satz gelesen werden soll

– Zeilennummer eines Satzes der Arbeitsdatei im Puffer EDTKEY1 (der Satz muss nicht existieren).

– Displacement n (0, +N, −N) im Feld EAMDISP (EDTAMCB): Abstand (in Sätzen) zur angegeben Zeilennummer im binären Format

– Zeichensatz, in dem der Satz bereitgestellt werden soll, im Feld EGLCCSN (EDTGLCB)

Sätze mit Satzmarkierung 13 (siehe Abschnitt „IEDTPTM - Markieren eines Satzes“ auf Seite 80) werden nur berücksichtigt, wenn im Kontrollblock EDTAMCB im Feld EAMFLAG das Kennzeichen EAMIGN13 gesetzt ist. Andernfalls werden sie behandelt, als wären sie nicht existent.

Die Schnittstelle erlaubt zwei Arten der Adressierung:

– Absolute Adressierung

– Relative Adressierung

Lesen eines Satzes mit einer bestimmten Zeilennummer - Absolute Adressierung

Angabe des Displacement (EAMDISP) : n = 0

Wird als Displacement der Wert 0 angegeben, dann wird nach dem Satz mit der angegebenen Zeilennummer (EDTKEY1) gesucht.

Ist dieser Satz nicht vorhanden, wird der Satz mit der nächsten Zeilennummer übergeben (evtl. der erste oder der letzte Satz).

Übersicht über Returncodes und gelesenen Satz:

EDTKEY1	EAMDISP	EGLMRET	EAMSR1	EDTREC
Zeilennummer eines existierenden Satzes	0	EAMRETOK	EAMOK00	Satz mit Zeilennummer = EDTKEY1
kleiner als Zeilennummer des ersten Satzes	0	EAMRETOK	EAMOK08	erster Satz
größer als Zeilennummer des letzten Satzes	0	EAMRETOK	EAMOK12	letzter Satz
größer als Zeilennummer des ersten Satzes und kleiner als Zeilennummer des letzten Satzes und keine Zeilennummer eines existierenden Satzes	0	EAMRETOK	EAMOK04	nächster Satz nach EDTKEY1
Arbeitsdatei enthält keine Sätze		EAMACERR	EAMAC16	

Der EDT übergibt im Erfolgsfall die Zeilennummer des tatsächlich gelesenen Satzes in EDTKEY.
Bedeutung der Returncodes siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

Lesen eines Satzes mit relativer Adressierung

Angabe des Displacement (EAMDISP): n = +N/−N (N≠0)

Die Adresse des zu lesenden Satzes setzt sich zusammen aus

– der Zeilennummer eines Satzes (EDTKEY1),

– dem Abstand N des Satzes zur angegebenen Zeilennummer:

  +N: der N-te (logische) Satz nach der angegebenen Zeilennummer wird gelesen,

  −N: der N-te (logische) Satz vor der angegebenen Zeilennummer wird gelesen.

Ist der gewünschte Satz außerhalb des Zeilennummernbereichs der Arbeitsdatei, wird der erste bzw. der letzte Satz geliefert.

Wird eine der Zeilennummern X'0000000000000000' bzw. X'FFFFFFFFFFFFFFFF' als Nummer des zu lesenden Satzes angegeben, wird die Abstandsangabe relativ zu einem fiktiven Satz vor allen anderen bzw. nach allen anderen berechnet. D.h. die Abstandsangabe 1 bzw. −1 liefert dann den ersten bzw. letzten Satz der Arbeitsdatei.

Übersicht über Returncodes und gelesenen Satz:

EDTKEY1	**EAMDISP**	**EGLMRET**	**EAMSR1**	**EDTREC**
EDTKEY1 = XXXXXXXX	+N (N <= Sätze nach EDTKEY1)	EAMRETOK	EAMOK00	Satz N nach EDTKEY1
EDTKEY1 = XXXXXXXX	−N (N <= Sätze vor EDTKEY1)	EAMRETOK	EAMOK08	Satz N vor EDTKEY1
EDTKEY1 = XXXXXXXX	+N (N > Sätze nach EDTKEY1)	EAMRETOK	EAMOK12	letzter Satz
EDTKEY1 = XXXXXXXX	−N (N > Sätze vor EDTKEY1)	EAMRETOK	EAMOK04	erster Satz
Arbeitsdatei enthält keine Sätze		EAMACERR	EAMAC16	

Der EDT übergibt im Erfolgsfall die Zeilennummer des tatsächlich gelesenen Satzes in EDTKEY.

Bedeutung der Returncodes siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

– Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTAMCB

– Versorgen des Puffers EDTKEY1

– Bereitstellen von Speicherplatz für die Puffer EDTKEY und EDTREC

– Aufruf der Einsprungadresse IEDTGET mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Einsprungadresse	:	IEDTGET

Parameterliste	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY, EDTREC)

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
	EGLCCSN		
EDTAMCB:	EAMUNIT	EDTAMCB:	EAMMARK
	EAMVERS		EAMLKEY
	EAMFILE		EAMLREC
	EAMDISP	EDTKEY	
	EAMLKEY1	EDTREC	
	EAMPKEY		
	EAMPREC		
EDTKEY1			
EDTKEY			
EDTREC			

Mögliche Returncodes siehe „Logische Satzzugriffsfunktionen“ auf Seite 66.

Rückkehrparameter bei erfolgreichem Satzzugriff

Neben dem Feld EGLRETC des EDTGLCB (EGLMRET = EAMRETOK) werden vom EDT folgende Parameter versorgt:

Satz	EDTREC
Satzlänge	EAMLREC im EDTAMCB
Zeilennummer	EDTKEY (immer 8 Byte lang)
Länge der Zeilenummer	EAMLKEY im EDTAMCB (immer 8 Byte)
Markierungen des Satzes	EAMMARK im EDTAMCB

Wenn die Pufferlänge (EAMPREC) nicht zur Aufnahme des Satzes ausreicht, werden EGLMRET mit EAMACERR und EGLSR1 mit EAMAC12 versorgt. Der Satz wird abgeschnitten. Das Feld EAMLREC im AMCB wird mit der tatsächlich gelesenen Länge versorgt.

Im Kompatibilitäts-Modus wird dort (wie in EDT V16.6) die für ein komplettes Lesen erforderliche Pufferlänge abgelegt. Im Unicode-Modus ist die Angabe der wirklich übertragenen Bytes jedoch erforderlich, weil bei Mehrbyte-Codierung das Abschneiden des Satzes grundsätzlich zwischen zwei gültigen Zeichen erfolgen muss und daher evtl. weniger Bytes übertragen werden, als in EAMPREC angegeben wurde.

Bei nicht erfolgreichem Zugriff werden die Felder EAMLKEY und EAMLREC mit dem Wert 0 versorgt.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Der Kontrollblock EDTAMCB wird folgendermaßen deklariert und initialisiert (der Wert 1024 ist hier als Beispiel für die maximal erwartete Satzlänge gewählt):

```
iedamcb amcb = IEDAMCB_INIT;
char rec[1024];
char key[8], key1[8];
amcb.length_key1 = 8;
amcb.length_key_outbuffer = 8;
amcb.length_rec_outbuffer = 1024;
```

Die Versorgung der weiteren Parameter ist benutzerabhängig. Wenn beispielsweise der 1. Satz in der Arbeitsdatei 0 gesucht wird:

```
strncpy(amcb.filename,"$0     ",8);
strncpy(key1,"00000001",8);
amcb.displacement = 0;
```

Aufruf der Funktion:

```
IEDTGET(&glcb,&amcb,key1,key,rec);
```

3.5.2 IEDTGTM - Lesen eines markierten Satzes

Diese Zugriffsfunktion bietet die Möglichkeit, ausgehend von einer bestimmten Zeilennummer (EDTKEY1), einen markierten Satz zu suchen und zu lesen.
Die Suchrichtung kann dabei bestimmt werden.

Es kann nur nach Sätzen gesucht werden, die markiert sind. Nach einer bestimmten Markierung kann nicht gesucht werden.

Gesucht werden kann in den Arbeitsdateien 0..22. In einer Arbeitsdatei, in der eine Datei real durch @OPEN (Format 2) geöffnet ist, können keine markierten Sätze gelesen werden. Der Zugriff wird mit einem Returncode abgewiesen.

Suchen des markierten Satzes

Für die Suche nach einem markierten Satz sind anzugeben:

– Arbeitsdatei, ($0..$22) im Feld EAMFILE (EDTAMCB):
Arbeitsdatei, aus der der markierte Satz gelesen werden soll.

– Zeilennummer eines Satzes der Datei im Puffer EDTKEY1 (der Satz muss nicht existieren)

– Displacement n (0,+1,-1) im Feld EAMDISP (EDTAMCB):
Angabe der Suchrichtung (andere positive Werte für Displacement werden wie +1, andere negative Werte werden wie -1 behandelt)

– Zeichensatz, in dem der Satz bereitgestellt werden soll, im Feld EGLCCSN (EDTGLCB)

Zwei Arten der Suche sind möglich:

– Suchen nach einer bestimmten Zeilennummer

– Suchen des nächsten markierten Satzes relativ zu einer angegebenen Zeilennummer

Lesen eines markierten Satzes mit einer bestimmten Zeilennummer

Angabe des Displacement n = 0

Wird als Displacement (EAMDISP) der Wert 0 angegebenen, wird der Satz mit der angegebenen Zeilennummer (EDTKEY1) gelesen. Ist dieser Satz nicht vorhanden oder enthält dieser Satz keine Markierungen, wird der nächste markierte Satz (eventuell der erste oder letzte markierte Satz) gelesen und übergeben.

Übersicht über Returncodes und gelesenen Satz:

EDTKEY1	EAMDISP	EGLMRET	EAMSR1	EDTREC
Zeilennummer eines existierenden Satzes	0	EAMRETOK	EAMOK00	Satz mit Zeilennummer
kleiner als Zeilennummer des ersten markierten Satzes	0	EAMRETOK	EAMOK08	erster Satz mit Markierung
größer als Zeilennummer des letzten markierten Satzes	0	EAMRETOK	EAMOK12	letzter Satz mit Markierung
größer als Zeilennummer des ersten markierten Satzes und kleiner als Zeilennummer des letzten markierten Satzes und nicht Zeilennummer eines markierten Satzes	0	EAMRETOK	EAMOK04	nächster markierter Satz nach EDTKEY1
Arbeitsdatei ist leer oder enthält keine markierten Sätze		EAMACERR	EAMAC16	

Der EDT übergibt im Erfolgsfall die Zeilennummer des tatsächlich gelesenen Satzes in EDTKEY und die Markierungen des Satzes in EAMMARK (EDTAMCB).

Bedeutung der Returncodes siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

Lesen des nächsten markierten Satzes

Angabe des Displacement n = +1 oder −1

Gesucht wird nach dem nächsten markierten Satz vor bzw. nach einer bestimmten Zeilennummer (EDTKEY1).

n = +1: der erste markierte Satz nach der angegebenen Zeilennummer wird gelesen
n = −1: der erste markierte Satz vor der angegebenen Zeilennummer wird gelesen

Ist in der angegebenen Richtung kein markierter Satz mehr vorhanden, wird der erste oder letzte markierte Satz gelesen und übergeben.

Übersicht über Returncodes und gelesenen Satz:

EDTKEY1	EAMDISP	EGLMRET	EAMSR1	EDTREC
EDTKEY1 = XXXXXXXX	+1 Satz mit Markierung existiert nach EDTKEY1	EAMRETOK	EAMOK00	nächster Satz mit Markierung nach EDTKEY1
EDTKEY1 = XXXXXXXX	−1 Satz mit Markierung existiert vor EDTKEY1	EAMRETOK	EAMOK00	nächster Satz mit Markierung vor EDTKEY1
EDTKEY1 = XXXXXXXX	+1 Kein Satz mit Markierung nach EDTKEY1	EAMRETOK	EAMOK12	letzter markierter Satz
EDTKEY1 = XXXXXXXX	−1 Kein Satz mit Markierung vor EDTKEY1	EAMRETOK	EAMOK08	erster markierter Satz
Arbeitsdatei ist leer oder enthält keine markierten Sätze		EAMACERR	EAMAC16	

Der EDT übergibt im Erfolgsfall die Zeilennummer des tatsächlich gelesenen Satzes in EDTKEY und die Markierungen des Satzes in EAMMARK (EDTAMCB).

Bedeutung der Returncodes siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

– Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTAMCB

– Versorgen des Puffers EDTKEY1

– Bereitstellen von Speicherplatz für die Puffer EDTKEY und EDTREC

– Aufruf der Einsprungadresse IEDTGTM mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.)

Einsprungadresse	:	IEDTGTM

Parameterliste	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY, EDTREC)

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
	EGLCCSN		
EDTAMCB:	EAMUNIT	EDTAMCB:	EAMMARK
	EAMVERS		EAMLKEY
	EAMFLAG		EAMLREC
	EAMFILE		
	EAMDISP	EDTKEY	
	EAMLKEY1	EDTREC	
	EAMPKEY		
	EAMPREC		
EDTKEY1			
EDTKEY			
EDTREC			

Rückkehrparameter bei erfolgreichem Satzzugriff:

Neben dem Feld EGLRETC des EDTGLCB (EGLMRET = EAMRETOK) werden vom EDT folgende Parameter versorgt:

Satz	EDTREC
Satzlänge	EAMLREC im EDTAMCB
Zeilennummer	EDTKEY (immer 8 Byte lang)
Länge der Zeilenummer	EAMLKEY im EDTAMCB (immer 8 Byte)
Markierungen des Satzes	EAMMARK im EDTAMCB

Wenn die Pufferlänge (EAMPREC) nicht zur Aufnahme des Satzes ausreicht, werden EGLMRET mit EAMACERR und EGLSR1 mit EAMAC12 versorgt. Der Satz wird abgeschnitten. Das Feld EAMLREC im AMCB wird mit der tatsächlich gelesenen Länge versorgt. Im Kompatibilitäts-Modus wird dort (wie in EDT V16.6B) die für ein komplettes Lesen erforderliche Pufferlänge abgelegt.

Im Unicode-Modus ist die Angabe der wirklich übertragenen Bytes jedoch erforderlich, weil bei Mehrbyte-Codierung das Abschneiden des Satzes grundsätzlich zwischen zwei gültigen Zeichen erfolgen muss und daher evtl. weniger Bytes übertragen werden, als in EAMPREC angegeben wurde.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Der Kontrollblock EDTAMCB wird folgendermaßen deklariert und initialisiert (der Wert 4096 ist hier als Beispiel für die maximal erwartete Satzlänge gewählt):

```
iedamcb amcb = IEDAMCB_INIT;
char key1[8];
char key[8];
char rec[4096];
ambc.length_key1 = 8;
amcb.length_key_outbuff = 8;
amcb.length_rec_outbuff = 4096;
```

Die Versorgung der weiteren Parameter ist benutzerabhängig. Wenn beispielsweise der nächste markierte Satz nach dem Satz mit der Zeilennummer 123.4 in der Arbeitsdatei 22 gesucht wird:

```
strncpy(amcb.filename,"$22     ",8);
strncpy(key1,"01234000",8);
amcb.displacement = 1;
```

Im C-Programm wird die Funktion IEDTGTM folgendermaßen aufgerufen:

```
IEDTGTM(&glcb,&amcb,key1,key,rec);
```

3.5.3 IEDTPUT - Schreiben eines Satzes

Diese Zugriffsfunktion speichert einen Satz (EDTREC) mit einer Markierung (EAMMARK im EDTAMCB) in einer Arbeitsdatei unter der angegebenen Zeilennummer (EDTKEY).

Ist bereits ein Satz mit dieser Zeilennummer vorhanden, wird er einschließlich seiner Markierungen (siehe Abschnitt „IEDTPTM - Markieren eines Satzes“ auf Seite 80) ersetzt.

Im Feld EGLCCSN (EDTGLCB) ist der Zeichensatz anzugeben, in dem der Satz bereitgestellt wird.

Ist beim Schreiben eines Satzes mit IEDTPUT das Kennzeichen EAMNOMOD im Kontrollblock EDTAMCB im Feld EAMFLAG gesetzt, so wird die Arbeitsdatei nicht als modifiziert gekennzeichnet, obwohl ein Satz aufgenommen wird. Dadurch erkennt das aufrufende Programm leichter, ob ein Anwender die Arbeitsdatei im Dialog geändert hat (durch Abfrage des Feldes EPLMODIF im Kontrollblock EDTPARL). Außerdem entfällt eine unnötige Sicherungsabfrage des EDT bei der Anweisung @HALT, wenn im Dialog nichts geändert wurde.

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

– Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTAMCB
– Versorgen des Puffers EDTKEY
– Versorgen des Puffers EDTREC
– Aufruf der Einsprungadresse IEDTPUT mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Einsprungadresse	:	IEDTPUT

Parameterliste	:	A (EDTGLCB, EDTAMCB, EDTKEY, EDTREC)

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
	EGLCCSN		
EDTAMCB:	EAMUNIT		
	EAMVERS		
	EAMFLAG		
	EAMFILE		
	EAMMARK		
	EAMLKEY		
	EAMLREC		
EDTKEY			
EDTREC			

Mögliche Returncodes siehe Abschnitt „Logische Satzzugriffsfunktionen“ auf Seite 66.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Der Kontrollblock EDTAMCB wird folgendermaßen deklariert und initialisiert (der Wert 4096 ist hier als Beispiel für die maximal erwartete Satzlänge gewählt):

```
iedamcb amcb = IEDAMCB_INIT;
char key[8];
char rec[4096];
IEDAMCB_SET_NO_MARKS(amcb);
amcb.length_key = 8;
```

Wenn beispielsweise ein Satz mit der Zeilennummer 75 in die Arbeitsdatei 0 geschrieben werden soll:

```
strncpy(rec,"Das ist der Inhalt des Satzes",29);
strncpy(key,"00750000",8);
strncpy(amcb.filename,"$0     ",8);
amcb.length_rec = 29;
```

Im C-Programm wird die Funktion IEDTPUT folgendermaßen aufgerufen:

```
IEDTPUT(&glcb,&amcb,key,rec);
```

3.5.4 IEDTPTM - Markieren eines Satzes

Mit dieser Zugriffsfunktion kann ein Satz in einer Arbeitsdatei markiert werden oder seine Markierung gelöscht werden.

In einer Arbeitsdatei, in der eine Datei real durch @OPEN (Format 2) geöffnet ist, können keine Sätze markiert werden.

Mögliche Satzmarkierungen

- Die Satzmarkierungen 1 bis 9 (EAMMK01 bis EAMMK09 in EDTAMCB) stehen dem Anwender frei zur Verfügung. Er kann diese Markierungen mit den Funktionen IEDTPUT (Schreiben Satz und Markierungen) und IEDTPTM (Schreiben Satzmarkierung) verändern.
  Diese Markierungen können auch durch (Kurz-)Anweisungen gesetzt oder gelöscht werden.
- Die Satzmarkierung 13 (EAMMK13 in EDTAMCB) hat die Sonderfunktion eines Ignorier-Indikators. Derart markierte Sätze werden
  - bei der Rückkehr aus dem (mit @DIALOG bzw. @EDIT ONLY eingeleiteten) Benutzerdialog (siehe Abschnitt „IEDTCMD - Ausführen von EDT-Anweisungen“ auf Seite 55) automatisch gelöscht
  - beim Schreiben in eine Datei bzw. ein Bibliothekselement nicht übernommen
  - beim Kopieren von Zeilen nicht kopiert
  - durch die Satzzugriffsfunktionen IEDTGET und IEDTPTM nur dann berücksichtigt, wenn im Kontrollblock EDTAMCB im Feld EAMFLAG das Kennzeichen EAMIGN13 gesetzt ist
- Die Satzmarkierung 14 (EAMMK14 in EDTAMCB) hat die Sonderfunktion eines Update-Indikators. Derart markierte Sätze sind im F-Modus überschreibbar, auch wenn dies nicht explizit durch den Dialog-Anwender verlangt wird.
- Die Satzmarkierung 15 (EAMMK15 in EDTAMCB) hat die Sonderfunktion eines Schreibschutzindikators. Sätze mit Satzmarkierung 15 sind schreibgeschützt. Sie können mit der Kurzanweisung X oder F2 im F-Modus-Bildschirmdialog nicht auf überschreibbar gestellt werden.

Voraussetzung für die Auswertung der Satzmarkierungen 14 und 15 durch den EDT ist die Einstellung von PROTECTION=ON mit der @PAR-Anweisung.

Die Satzmarkierungen 13, 14 und 15 können nur durch die Satzzugriffsfunktionen IEDTPUT und IEDTPTM, nicht aber durch EDT-Anweisungen verändert werden.
Durch Ändern des Satzes im Dialog (d.h. Eingabe von Daten im Datenfenster) werden die Satzmarkierungen 13, 14 und 15 gelöscht.

Die Zeilennummer des zu markierenden Satzes muss im Feld EDTKEY angegeben werden.

Das Setzen der Markierung wird als logisches ODER realisiert. Sind alle Bits (auch die undefinierten) in EAMMMARK gleich Null, wird die Markierung gelöscht.

Existiert kein Satz mit der angegebenen Satznummer, wird der Aufruf mit Returncode abgewiesen.

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

– Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTAMCB

– Versorgen des Puffers EDTKEY

– Aufruf der Einsprungadresse IEDTPTM mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Einsprungadresse	:	IEDTPTM

Parameterliste	:	A (EDTGLCB, EDTAMCB, EDTKEY)

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
EDTAMCB:	EAMUNIT		
	EAMVERS		
	EAMFLAG		
	EAMFILE		
	EAMMARK		
	EAMLKEY		
EDTKEY			

Mögliche Returncodes siehe Abschnitt „Logische Satzzugriffsfunktionen“ auf Seite 66.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Der Kontrollblock EDTAMCB wird folgendermaßen deklariert und initialisiert:

```
iedamcb amcb = IEDAMCB_INIT;
char key[8];
IEDAMCB_SET_NO_MARKS(amcb);
amcb.length_key = 8;
```

Wenn beispielsweise ein Satz mit der Zeilennummer 123.4 in der Arbeitsdatei 1 im F-Modus überschreibbar dargestellt werden soll (Satzmarkierung 14):

```
strncpy(amcb.filename,"$1     ",8);
strncpy(key,"01234000",8);
MARK_14(amcb) = 1;
```

Im C-Programm wird die Funktion IEDTPTM folgendermaßen aufgerufen:

```
IEDTPTM(&glcb,&amcb,key);
```

3.5.5 IEDTDEL - Löschen eines Satzbereiches oder des Kopierpuffers

Durch diese Zugriffsfunktion wird der angegebene Satzbereich einer bestehenden Arbeitsdatei oder der Kopierpuffer gelöscht.

Das Löschen eines Satzbereichs entspricht der Anweisung @DELETE (Format 1).

Das Löschen des Kopierpuffers entspricht der Kurzanweisung * im F-Modus.

Löschen eines Satzbereichs

Der Satzbereich wird durch zwei Zeilennummern angegeben:

EDTKEY1: Zeilennummer des ersten Satzes des Bereiches

EDTKEY2: Zeilennummer des letzten Satzes des Bereiches

Wird in EDTKEY1 der Wert X'0000000000000000' und in EDTKEY2 der Wert X'FFFFFFFFFFFFFFFF' angegeben, werden alle Sätze der Arbeitsdatei gelöscht. Dies entspricht der Anweisung @DELETE % - . $ (nicht der Anweisung @DEL ohne Operanden, die alle Werte der Arbeitsdatei zurücksetzt).

Löschen des Kopierpuffers

Im Kontrollblock EDTAMCB im Feld EAMFILE muss der Wert C abgelegt werden.

Der Wert muss linksbündig stehen und mit Leerzeichen aufgefüllt sein.

Die Puffer EDTKEY1 und EDTKEY2 müssen beim Löschen des Kopierpuffers nicht angegeben werden.

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

– Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTAMCB
– Versorgen des Puffers EDTKEY1
– Versorgen des Puffers EDTKEY2
– Aufruf der Einsprungadresse IEDTDEL mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Einsprungadresse	:	IEDTDEL

Parameterliste	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY2)

Aufrufparameter		Rückkehrparameter	
EDTAMCB:	EAMFILE EAMLKEY1 EAMLKEY2	EDTGLCB:	EGLRETC EGLRMSG
EDTKEY1 EDTKEY2			

Mögliche Returncodes siehe Abschnitt „Logische Satzzugriffsfunktionen“ auf Seite 66.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Der Kontrollblock EDTAMCB wird folgendermaßen deklariert und initialisiert:

```
iedamcb amcb = IEDAMCB_INIT;
char key1[] = "08000001";
char key2[] = "99999999";
amcb.length_key1 = amcb.length_key2 = 8;
```

Wenn beispielsweise die Sätze von der Zeilennummer 800.0001 bis zum Ende der Arbeitsdatei 1 gelöscht werden sollen:

```
strncpy(amcb.filename,"$1      ",8);
```

Im C-Programm wird die Funktion IEDTDEL folgendermaßen aufgerufen:

```
IEDTDEL(&glcb,&tamcb,key1,key2);
```

3.5.6 IEDTREN - Ändern der Zeilennummer

Mit dieser Zugriffsfunktion wird die Zeilennummer eines Satzes in einer Arbeitsdatei geändert.

EDTKEY1: Zeilennummer, die geändert werden soll

EDTKEY2: neue Zeilennummer

Existiert bereits ein Satz mit der neuen Zeilennummer oder enthält die Arbeitsdatei zwischen der alten und der neuen Zeilennummer weitere Sätze, wird die Funktion nicht ausgeführt.

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

- Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTAMCB
- Versorgen des Puffers EDTKEY1
- Versorgen des Puffers EDTKEY2
- Aufruf der Einsprungadresse IEDTREN mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Einsprungadresse	:	IEDTREN

Parameterliste	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY2)

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
EDTAMCB:	EAMUNIT		
	EAMVERS		
	EAMFILE		
	EAMLKEY1		
	EAMLKEY2		
EDTKEY1			
EDTKEY2			

Mögliche Returncodes siehe Abschnitt „Logische Satzzugriffsfunktionen“ auf Seite 66.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Der Kontrollblock EDTAMCB wird folgendermaßen deklariert und initialisiert:

```
iedamcb amcb = IEDAMCB_INIT;
char key1[] = "01234000";
char key2[] = "00000100";
amcb.length_key1 = amcb.length_key2 = 8;
```

Wenn beispielsweise der Satz mit der Zeilennummer 123.4 der 1. Satz in der Arbeitsdatei 1 ist und die Zeilennummer 0.01 erhalten soll::

```
strncpy(amcb.filename,"$1      ",8);
```

Im C-Programm wird die Funktion IEDTREN folgendermaßen aufgerufen:

```
IEDTREN(&glcb,&amcb,key1,key2);
```

3.5.7 IEDTGET - Lesen der globalen Einstellungen

Mit der Funktion IEDTGET können Informationen über die globalen Einstellungen des EDT gelesen werden. Dazu muss im Feld EAMFILE im Kontrollblock EDTAMCB der Wert G angegeben werden (linksbündig und mit Leerzeichen aufgefüllt).

Bei Verwendung des V17-Formats des Kontrollblocks muss die Länge des Kontrollblocks EDTPARG im Feld EAMPREC (EDTAMCB) nicht mehr eingetragen werden, da die Länge der Ausgabe durch die Version des EDTPARG eindeutig bestimmt ist. Ebenso wird bei Rückkehr EAMLREC nicht mehr versorgt.

Die Informationen werden vom EDT im Kontrollblock EDTPARG abgelegt (siehe Abschnitt „EDTPARG - Globale Einstellungen“ auf Seite 41).

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

– Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTAMCB

– Angabe des *initialisierten* Kontrollblocks EDTPARG als Ausgabebereich

– Aufruf der Einsprungadresse IEDTGET mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Einsprungadresse	:	IEDTGET

Parameterliste	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY, EDTPARG)

Für die Puffer EDTKEY1 und EDTKEY können beliebige Werte angegeben werden, sie werden nicht ausgewertet.

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
EDTAMCB:	EAMUNIT	EDTPARG	
	EAMVERS		
	EAMFILE		
EDTPARG:	EPGUNIT		
	EPGVERS		

Returncodes bei der Statusabfrage

EGLMRET	EGLSR1
EAMRETOK	EAMOK00
EAMACERR	EAMAC12
EAMPAERR	EAMPA04
EAMPAERR	EAMPA08
EAMPAERR	EAMPA12
EAMPAERR	EAMPA32
EAMPAERR	EAMPA36

Nach erfolgreichem Aufruf (Returncode ist EAMRETOK/EAMOK00) sind die Informationen im Kontrollblock EDTPARG abgelegt.

Bedeutung der Returncodes siehe Abschnitt Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

War der Aufruf nicht erfolgreich, bleibt der Kontrollblock EDTPARG unverändert.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Die Kontrollblöcke EDTAMCB und EDTPARG werden folgendermaßen deklariert und initialisiert:

```
iedamcb amcb = IEDAMCB_INIT;
iedparg parg = IEDPARG_INIT;
strncpy(amcb.filename,"G       ", 8);
```

Im C-Programm wird die Funktion IEDTGET zum Lesen der globalen Einstellungen folgendermaßen aufgerufen:

```
IEDTGET(&glcb,&amcb,NULL,NULL,&parg);
```

3.5.8 IEDTGET - Lesen der arbeitsspezifischen Einstellungen

Mit der Funktion IEDTGET können spezifische Einstellungen einer Arbeitsdatei gelesen werden. Im Feld EAMFILE im Kontrollblock EDTAMCB muss einer der folgenden Werte abgelegt werden:

L0..L22 (Arbeitsdatei 0..22)

Die Werte müssen linksbündig stehen und mit Leerzeichen aufgefüllt sein.
Die Länge des Kontrollblocks EDTPARL muss nicht mehr im Feld EAMPREC (EDTAMCB) eingetragen werden, da die Länge der Ausgabe durch die Version des EDTPARL eindeutig bestimmt ist. Ebenso wird bei Rückkehr EAMLREC nicht mehr versorgt.
Die Informationen werden vom EDT im Kontrollblock EDTPARL abgelegt (siehe Abschnitt „EDTPARL - Arbeitsdateispezifische Einstellungen“ auf Seite 44).

Aufruf

Folgende Angaben sind notwendig (siehe Übersichtstabelle):

- Versorgen der benötigten Felder in den Kontrollblöcken EDTGLCB und EDTAMCB
- Angabe des initialisierten Kontrollblocks EDTPARL als Ausgabebereich
- Aufruf der Einsprungadresse IEDTGET mit der Parameterliste

Übersichtstabelle

(Kontrollblöcke siehe Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22).

Einsprungadresse	:	IEDTGET

Parameterliste	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY, EDTPARL)

Für die Puffer EDTKEY1 und EDTKEY können beliebige Werte angegeben werden, sie werden nicht ausgewertet.

Aufrufparameter		Rückkehrparameter	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLRMSG
EDTAMCB:	EAMUNIT		
	EAMVERS	EDTPARL	
	EAMFILE		
EDTPARL:	EPLUNIT		
	EPLVERS		

Returncodes bei der Statusabfrage

EGLMRET	EGLSR1
EAMRETOK	EAMOK00
EAMACERR	EAMAC12
EAMPAERR	EAMPA04
EAMPAERR	EAMPA08
EAMPAERR	EAMPA12
EAMPAERR	EAMPA32
EAMPAERR	EAMPA36

Nach erfolgreichem Aufruf (Returncode ist EAMRETOK/EAMOK00) sind die Informationen im Kontrollblock EDTPARL abgelegt.

Bedeutung der Returncodes siehe Abschnitt Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23.

War der Aufruf nicht erfolgreich, bleibt der Kontrollblock EDTPARL unverändert.

Aufruf im C-Programm

Benötigte Include-Dateien:

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

Der Kontrollblock EDTAMCB wird folgendermaßen deklariert und initialisiert:

```
iedamcb amcb = IEDAMCB_INIT;
iedparl parl = IEDPARL_INIT;
strncpy(amcb.filename,"L1      ",8);
```

Im C-Programm wird die Funktion IEDTGET zum Lesen der arbeitsspezifischen Einstellungen folgendermaßen aufgerufen:

```
IEDTGET(&glcb,&amcb,NULL,NULL,&parl);
```

4 Benutzerdefinierte Anweisungen - @USE

Der EDT bietet die Möglichkeit, eigene Anweisungen zu schreiben. Dazu muss die Funktion der Anweisung in Form einer externen Routine (Anweisungsroutine) realisiert werden. Diese Routine ist üblicherweise als Modul in einer Bibliothek abgelegt. Es ist jedoch auch möglich, Anweisungsroutinen direkt in ein Hauptprogramm zu intergrieren, das dann den EDT über die Unterprogramm-Schnittstelle startet.

Eine Anweisungsroutine wird mit der Anweisung @USE definiert und kann dann über das bei @USE vereinbarte Benutzeranweisungssymbol aufgerufen werden.

Folgt der benutzerdefinierten Anweisung ein Text, wird er der Anweisungsroutine übergeben. Beim Rücksprung kann die Anweisungsroutine dem EDT eine Meldung übergeben.

4.1 Vereinbaren einer benutzerdefinierten Anweisung

Mit der Anweisung @USE wird eine benutzerdefinierte Anweisung vereinbart.
Die detaillierte Darstellung und Beschreibung der @USE-Anweisung findet sich im Handbuch Anweisungen [1].

@USE COMMAND = 'spec' ,ENTRY = entry, MODLIB = modlib

oder im kompatiblen Format:

@USE COMMAND = 'spec' (name[,modlib])

4.2 Aufruf einer benutzerdefinierten Anweisung

Durch Eingabe des definierten Benutzeranweisungssymbols (`spec`) für eine benuzerdefinierte Anweisung wird diese als Unterprogramm gestartet.

Der Anweisungsroutine kann ein beliebiger Text übergeben werden.

Wurde beim Vereinbaren mit @USE ein Einsprungpunkt festgelegt (@USE COMMAND= …,ENTRY=entry,…), wird bei Eingabe der benutzerdefinierten Anweisung der gesamte Text hinter dem Benutzeranweisungssymbol an die Anweisungsroutine übergeben. Dieser Text kann von der Anweisungsroutine beliebig interpretiert werden.

Wurde beim Vereinbaren mit @USE kein Einsprungpunkt festgelegt (@USE COMMAND=…,ENTRY=*,…), muss bei Eingabe der benutzerdefinierten Anweisung der Name des Einsprungpunktes (`entry`) angegeben werden. Dahinter kann eine Zeichenfolge angegeben werden, die der Anweisungsroutine übergeben wird.

```
spec entry [chars]
```

Dabei kann `entry` als Anweisungsname, `chars` als Operanden angesehen werden. Wurde das kompatible Format der @USE-Anweisung verwendet, werden maximal die ersten 8 Zeichen (in Großbuchstaben des Zeichensatzes `EDF03IRV` umgewandelt) der benutzerdefinierten Anweisung als Name des Einsprungpunktes betrachtet.

Wurde das neue Format verwendet, werden maximal die ersten 32 Zeichen (in Großbuchstaben des Zeichensatzes `EDF03IRV` umgewandelt) als Name des Einsprungpunktes betrachtet.

Einsprungpunkte, die auf diese Weise in der benutzerdefinierten Anweisung selbst angegeben werden, dürfen daher keine Kleinbuchstaben enthalten.

Es wird empfohlen, den Anweisungsnamen innerhalb derartiger benutzerdefinierter Anweisungen vom Rest der Anweisung durch ein Leerzeichen abzutrennen, um Fehlinterpretationen aufgrund des Abschneidens nach 8 bzw. 32 Zeichen auszuschließen.

Kann der an die Anweisungsroutine zu übergebende Text nicht in den Zeichensatz umgewandelt werden, der über die Initialisierungsroutine (siehe „Aufruf der Initialisierungsroutine zu einer benutzerdefinierten Anweisung“ auf Seite 96) festgelegt wurde, wird die Anweisung mit einer Fehlermeldung abgewiesen.

Ist das Benutzeranweisungssymbol gleich dem aktuellen EDT-Anweisungssymbol, hat die EDT-Anweisung Vorrang. Das heißt, nur wenn keine EDT-Anweisung (in irgendeiner zulässigen Abkürzung) identifiziert werden kann, wird angenommen, es handle sich um eine Benutzeranweisung. Es liegt in der Verantwortung des Benutzers, Konflikte zu vermeiden.

Beim Sprung in die Anweisungsroutine sind die Register wie folgt geladen:

Register	Datenbereich
(R1)	A (PARAMETERLISTE)
(R13)	A (SAVEAREA)
(R14)	A (RETURN)
(R15)	V (ENTRY)

Erläuterung der Registerinhalte siehe in Abschnitt „Aufruf des EDT“ auf Seite 20).

Parameterliste	:	A (EDTGLCB, COMMAND)

Folgende Parameter werden vom EDT an die Anweisungsroutine übergeben:

– **EDTGLCB**
Globaler EDT-Kontrollblock, wird vom EDT bereitgestellt und ist bereits initialisiert. Der Kontrollblock darf nach Verlassen der Anweisungsroutine nicht mehr verwendet werden.

Welche Version des Kontrollblocks `EDTGLCB` verwendet wird, wurde über die Initialisierungsroutine (siehe Abschnitt „Aufruf der Initialisierungsroutine zu einer benutzerdefinierten Anweisung“ auf Seite 96) festgelegt. Im Feld `EGLCDS` übergibt der EDT die Taste mit der die Anweisung abgeschickt wurde (nur bei Anweisungsfiltern, siehe Abschnitt „Spezialanwendung als Anweisungsfilter“ auf Seite 99).

Der Kontrollblock ist im Abschnitt „EDTGLCB - Globaler EDT - Kontrollblock“ auf Seite 23 beschrieben.

– **COMMAND**
Vom EDT bereitgestellter Puffer, der die Zeichenfolge enthält, die bei der Eingabe der externen Anweisung angegeben wurde.

Das Benutzeranweisungssymbol wird nicht übertragen. Die maximale Länge beträgt 32763 Byte + 4 Byte Längenfeld (siehe Abschnitt „Puffer“ auf Seite 50).

Der Puffer `COMMAND` ist in dem Zeichensatz codiert, der über die Initialisierungsroutine (siehe Aufruf der Initialisierungsroutine zu einer benutzerdefinierten Anweisung) vereinbart wurde.

Das Feld `EGLCCSN` wird vom EDT mit dem Namen des Zeichensatzes versorgt.
In Abhängigkeit von der Einstellung bei @PAR LOW wird die Zeichenfolge vor Übergabe an die Anweisungsroutine ggf. in Großbuchstaben umgewandelt.

Hinweis

Die weitere Beschreibung bezieht sich auf einen Aufruf mit Version 2 des Kontrollblocks `EDTGLCB`. Die Beschreibung für den Aufruf mit Version 1 des Kontrollblocks findet sich in [3].

Eine Anweisungsroutine sollte immer die Version des erhaltenen Kontrollblocks `EDTGLCB` abfragen.

Übersichtstabelle

Die Anweisungsroutine wird vom EDT mit folgender Parameterliste aufgerufen:

Parameterliste : A (EDTGLCB, COMMAND)

Aufrufparameter	Rückkehrparameter
werden vom EDT vor Aufruf der externen Anweisungsroutine versorgt	werden vom EDT nach Rückkehr aus der externen Anweidungsroutine ausgewertet
EDTGLCB: EGLUNIT EGLVERS EGLCCSN EGLCDS EGLUSR1 EGLUSR2 COMMAND	EDTGLCB: EGLRETC EGLRMSG EGLCMDS EGLFILE EGLUSR2

Returncodes bei benutzerdefinierten Anweisungen

Die Returncodes werden von der Anweisungsroutine gesetzt und vom EDT ausgewertet. Hat die Anweisungsroutine im Feld EGLRMSG keinen Meldungstext abgelegt, werden vom EDT je nach Returncode folgende Meldungen ausgegeben:

EGLMRET	EGLSR1	Bedeutung
EUPRETOK	00	Keine Meldung
EUPSYERR	00	Meldung EDT3991
EUPRTERR	00	Meldung EDT5991

Hat die Anweisungsroutine im Feld EGLRMSG eine eigene Meldung abgelegt, gibt der EDT *diese* Meldung anstelle der oben beschriebenen Meldungen aus (auch bei EUPRETOK). Der Meldungstext der Anweisungsroutine wird dabei je nach Returncode in eine der EDT-Meldungen EDT0999, EDT3999 bzw. EDT5999 eingesetzt.

Bei anderen Returncodes gibt der EDT die Meldung EDT5410 aus. Die eigene Meldung im Feld EGLRMSG wird in diesem Fall nicht ausgegeben.

Normalerweise wird die Anweisungsroutine mit der aufrufenden Instanz des EDT kommunizieren wollen, d.h. sie benutzt den an der Schnittstelle übergebenen EDTGLCB für Aufrufe der EDT-Unterprogrammschnittstelle.

Dabei stehen der Anweisungsroutine zur Verarbeitung folgende Funktionen zur Verfügung:

– die Satzzugriffsfunktionen `IEDTGET`, `IEDTPUT`, usw. (siehe Abschnitt „Logische Satzzugriffsfunktionen“ auf Seite 66).

– die Funktion `IEDTEXE` zum Ausführen einer EDT-Anweisung (siehe Abschnitt „IEDTEXE - Ausführen von EDT-Anweisungen ohne Bildschirmdialog“ auf Seite 62).

  Folgende Anweisungen dürfen bei `IEDTEXE` nicht angegeben werden:

  – eine weitere benutzerdefinierte Anweisung

  – der Aufruf einer Anwenderroutine (@RUN)

  – die Anweisungen @DIALOG, @EDIT, @END, @HALT, @MODE und @RETURN sowie

  – die Prozeduraufrufe @DO und @INPUT.

In Ausnahmefällen kann es auch gewünscht sein, dass die Anweisungsroutine mit einer andern Instanz des EDT kommuniziert. Dazu muss ein `EDTGLCB` benutzt werden, der unabhängig von dem mitgelieferten `EDTGLCB` initialisiert wurde oder noch gar nicht initialisiert ist. Da die beiden EDT-Instanzen vollkommen unabhängig voneinander laufen, bestehen für die Aufrufe an die andere Instanz keinerlei Einschränkungen. Es kann auf diesem Weg dann auch nicht auf die Daten der aufrufenden Instanz zugegriffen werden.

4.3 Aufruf der Initialisierungsroutine zu einer benutzerdefinierten Anweisung

Soll der EDT die Anweisungsroutine einer benutzerdefinierten Anweisung mit dem V17-Format der Schnittstelle aufrufen, muss der Ersteller der Anweisungsroutine zusätzlich eine Initialisierungsroutine bereitstellen.

Der Name des Einsprungpunktes der Initialisierungsroutine ergibt sich aus dem Namen der Anweisungsroutine, indem die Zeichenfolge `@I` angehängt wird.

`Beispiel`

Einsprungpunkt der Anweisungsroutine: `SELECT`

Einsprungpunkt der zugehörigen Initialisierungsroutine: `SELECT@I`

Die Initialisierungsroutine muss im gleichen Modul liegen wie die Anweisungsroutine, d.h. beim Laden der Anweisungsroutine muss die Initialisierungsroutine mitgeladen werden.

In der Initialisierungsroutine kann der Anwender den Zeichensatz festlegen, in dem der Puffer beim Aufruf der Anweisungsroutine codiert sein soll (im Feld `EGLCCSN` im Kontrollblock `EDTGLCB`). Lässt der Anwender das Feld `EGLCCSN` leer (Leerzeichen oder binär Null), nimmt der EDT den Zeichensatz `UTFE` an.
Darüber hinaus kann die Initialisierungsroutine spezifische Initialisierungen für die Anweisungsroutine vornehmen. Die Initialisierungsroutine kann die Funktionen der `IEDTGLE`-Schnittstelle benutzen. Dabei gelten die gleichen Einschränkungen wie bei Anweisungsroutinen.
Für jede Anweisungsroutine wird die Initialisierungsroutine immer dann aufgerufen, wenn die Anweisungsroutine geladen bzw. ihre Adresse lokalisiert wurde. Dabei wird jeder Einsprungpunkt als eigene Anweisungsroutine betrachtet (auch wenn mehrere Einsprungpunkte die gleiche Codestelle bezeichnen).

- Ist bei der Anweisung @USE kein fester Einsprungpunkt angegeben (als Einsprungpunkt ist `*` angegeben), ergibt sich der Name des Einsprungpunktes erst bei der Eingabe der benutzerdefinierten Anweisung. Die Initialisierungsroutine wird dann bei jedem Aufruf einer Benutzeranweisung mit diesem Einsprungpunkt gerufen. Folglich muss eine Initialisierungsroutine für derartige Anweisungen so geschrieben sein, dass sie mit mehrfachen Aufrufen umgehen kann.
- Ist bei der Anweisung @USE ein fester Einsprungpunkt angegeben, wird die Anweisungsroutine bei der @USE-Anweisung geladen und die Initialisierungsroutine unmittelbar nach dem Laden gerufen. Dies geschieht in der Regel nur einmal. Wird das Benutzersymbol zwischenzeitlich jedoch einer anderen Anweisungsroutine zugeordnet, wird die Initialisierungsroutine beim erneuten Zuweisen der ersten Anweisungsroutine auch erneut aufgerufen, unabhängig davon, ob die erste Anweisungsroutine neu geladen werden musste oder nicht.

Hat der Ersteller keine Initialisierungsroutine definiert, wird die Anweisungsroutine mit der Version 1 des Kontrollblocks EDTGLCB gerufen (V16-Format). Der Puffer COMMAND wird dann immer im Zeichensatz UTFE codiert.

Beim Sprung in die Initialisierungsroutine sind die Register wie folgt geladen:

Register	Datenbereich
(R1)	A (PARAMETERLISTE)
(R13)	A (SAVEAREA)
(R14)	A (RETURN)
(R15)	V (ENTRY)

Erläuterung Registerinhalte siehe Abschnitt „Aufruf des EDT“ auf Seite 20.

Die Initialisierungsroutine wird vom EDT mit folgender Parameterliste aufgerufen::

Parameterliste	:	A (EDTGLCB)

Der Kontrollblock EDTGLCB, wird vom EDT bereitgestellt und ist bereits initialisiert. Der Kontrollblock darf nach Verlassen der Initialisierungsroutine nicht mehr verwendet werden. Die Initialisierungsroutine wird mit der Version 2 des Kontrollblocks gerufen.

Aufrufparameter		Rückkehrparameter	
werden vom EDT vor Aufruf der Initialisierungsroutine versorgt		können von der Initialisierungsroutine versorgt werden und werden vom EDT nach der Rückkehr ausgewertet	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		EGLUSR2
	EGLUSR1		EGLCCSN
	EGLUSR2		EGLINDB

Returncodes bei Initialisierungsroutinen

Die Returncodes müssen von der Initialisierungsroutine gesetzt werden und werden vom EDT ausgewertet.

EGLMRET	EGLSR1	Bedeutung
EUPRETOK	00	Kein Fehler
EUPVEERR	00	Version wird von der Initialisierungsroutine nicht unterstützt; die benutzerdefinierte Anweisung wird mit der Meldung EDT5470 abgewiesen.
EUPRTERR	00	Sonstige Fehler. Die benutzerdefinierte Anweisung wird mit der Meldung EDT5471 abgewiesen.

Nach der Rückkehr aus der Initialisierungsroutine wertet der EDT das Feld EGLCCSN aus. Bei allen Aufrufen der zugehörigen Anweisungsroutine wird der Puffer COMMAND in diesem Zeichensatz codiert. Dabei legt der in der Initialisierungsroutine vereinbarte Zeichensatz nur fest, in welchem Zeichensatz der Puffer an die Anweisungsroutine übergeben wird. Beim Aufruf von EDT-Schnittstellen durch die Anweisungsroutine kann der Aufrufer selbstverständlich durch Überschreiben des Feldes EGLCCSN einen anderen Zeichensatz festlegen. Dieses Überschreiben hat allerdings keine Auswirkungen auf nachfolgende Aufrufe der Anweisungsroutine sondern gilt nur lokal für die aufgerufene IEDTGLE-Schnittstelle.

Die Aufrufe der Anweisungsroutine erfolgen mit der Version 2 des Kontrollblocks EDTGLCB.

Das Flag EGLCOMP im Indikator-Byte EGLINDB wird nach Rückkehr aus der Initialisierungsroutine ebenfalls ausgewertet. Wenn die Initialisierungsroutine dieses Feld gesetzt hat, erhält ein Anweisungsfilter (siehe Abschnitt „Spezialanwendung als Anweisungsfilter“ auf Seite 99) die zu filternde Anweisung grundsätzlich in Großbuchstaben, wurde es nicht gesetzt, wird je nach Einstellung bei @PAR LOW die Anweisung in Großbuchstaben oder in Groß-/Kleinschreibung übergeben.

Läuft die Version EDT V16.6B oder EDT V17.0A im Kompatibilitätsmodus, wird die Initialisierungsroutine nicht gerufen. Der Aufruf der Anweisungsroutine erfolgt mit der Version 1 des Kontrollblocks EDTGLCB.

Eine Anweisungsroutine, die sowohl im Unicode-Modus als auch im Kompatibilitäts-Modus bzw. mit EDT V16.6B laufen soll, muss die Version (EGLVERS) des vom EDT übergebenen Kontrollblocks abfragen und entsprechend darauf reagieren.

4.4 Spezialanwendung als Anweisungsfilter

Bei Aufruf des EDT über die Funktion IEDTCMD oder IEDTEXE (siehe Abschnitt „IEDTCMD - Ausführen von EDT-Anweisungen“ auf Seite 55) hat das rufende Programm die Möglichkeit, eine benutzerdefinierte Anweisung mit leerem Benutzeranweisungssymbol als Anweisungsfilter zu vereinbaren:

@USE COMMAND = ' ' ,ENTRY=entry,MODLIB=modlib

oder im kompatiblen Format:

@USE COMMAND=' ' (name [,modlib])

Diese Routine erhält dann jede Anweisung, die

– im F-Modus in der Anweisungszeile
– im L-Modus oder
– in EDT-Prozeduren

eingegeben wurde.

Die Routine erhält nicht

– Kurzanweisungen, die im F-Modus eingegeben wurden,
– Anweisungen, die über die Programmschnittstellen IEDTCMD oder IEDTEXE eingegeben wurden.

Die Routine erhält eine Anweisung auch dann, wenn der Anweisungsname unbekannt ist oder die Anweisung Syntaxfehler enthält.
Werden mehrere Anweisungen (mit Semikolon getrennt) eingegeben, werden die einzelnen Anweisungen nacheinander an die Routine übergeben.
Das Anweisungssymbol selbst wird nur dann übertragen, wenn es sich um eine Benutzeranweisung handelt. Führende Leerzeichen vor dem Anweisungssymbol werden nicht übertragen.

Im Feld EGLCDS (EDTGLCB) übergibt der EDT die Taste, mit der die Anweisung abgeschickt wurde.

Kann eine Anweisung nicht in den Zeichensatz umgewandelt werden, der für den Anweisungsfilter in der Initialisierungsroutine (siehe Abschnitt „Aufruf der Initialisierungsroutine zu einer benutzerdefinierten Anweisung“ auf Seite 96) vereinbart wurde, wird die Anweisung mit einer Fehlermeldung abgewiesen. Daher sollte ein Anweisungsfilter möglichst in einem Unicode-Zeichensatz arbeiten (empfohlen wird UTFE).

Anweisungsfilter können in ihrer Initialisierungsroutine auch festlegen, ob sie die Anweisung grundsätzlich in Großbuchstaben oder in Groß-/Kleinschreibung erwarten. Im ersten Fall ist durch die Initialisierungsroutine das Flag EGLCOMP (EDTGLCB) im EGLINDB zu setzen (siehe auch Abschnitt „Aufruf der Initialisierungsroutine zu einer benutzerdefinierten Anweisung“ auf Seite 96).

Ein Anweisungsfilter kann die Funktionen der IEDTGLE-Schnittstelle benutzen. Dabei gelten die gleichen Einschränkungen wie bei Anweisungsroutinen.

In einem Anweisungsfilter kann die vom EDT übergebene Anweisung nicht verändert werden (d.h. alle Änderungen werden ignoriert). Die Anweisungsroutine kann aber bei der Rückkehr zum EDT im Returncode EGLSR1 folgende Werte übergeben:
.

EGLMRET	EGLSR1	Bedeutung
EUPRETOK	EUPOK00	Die Anweisung soll ausgeführt werden.
EUPRETOK	EUPOK12	EDT soll den laufenden Anweisungsdialog beenden. Es wird zum Hauptprogramm zurückgekehrt als ob @HALT eingegeben worden wäre.
EUPRETOK	EUPOK24	Die Anweisung soll nicht ausgeführt werden.

Hinweis

Der EDT V17.0A unterstützt im Unicode-Modus die Implementierung von Anweisungs- und Anwenderroutinen in C (einschließlich der korrekten Versorgung des C-Laufzeitsystems).
Man beachte jedoch, dass derartige Routinen nicht mit einem EDT V16.6B ablaufen können, da dieser die Anweisungsroutine nicht mit einer für das Laufzeitsystem korrekt versorgten Savearea aufruft. Im EDT V17.0A, Kompatibilitäts-Modus, wurde diese Schwachstelle beseitigt. Das Laufzeitsystem kann jedoch im Kompatibilitäts-Modus nur dann korrekt versorgt werden, wenn die Anweisungsroutine dynamisch aus einer Bibliothek geladen wird. Soll eine in C geschriebene Anweisungsroutine also sowohl im Unicode wie im Kompatibilitäts-Modus laufen, sollte sie nicht statisch zu einem Hauptprogramm gebunden werden.

5 Anwenderroutinen - @RUN

Mit der Anweisung @RUN wird eine Anwenderroutine als Unterprogramm gestartet. Die Beschreibung der @RUN-Anweisung findet sich im Handbuch Anweisungen[1].

@RUN ENTRY = ... [,MODLIB =...] [,UNLOAD] [, '...']

Im Unicode-Modus wird die Anwenderroutine mit der gleichen Schnittstelle gerufen, mit der auch die Anweisungsroutine einer benutzerdefinierten Anweisung gerufen wird (siehe Kapitel „Benutzerdefinierte Anweisungen - @USE“ auf Seite 91). Die in der Anweisung angegebene Zeichenfolge wird im Puffer `COMMAND` übergeben.

Der einzige Unterschied zur Anweisungsroutine besteht darin, dass die EDT-Fehlerbehandlung bei Aufruf einer Anwenderroutine abgeschaltet wird, während sie bei Anweisungsroutinen eingeschaltet bleibt (sofern sie es vorher war).

Die Anwenderroutine kann die Funktionen der `IEDTGLE`-Schnittstelle benutzen. Dabei gelten die gleichen Einschränkungen wie bei Anweisungsroutinen.

Auch für eine Anwenderroutine wird (bei jedem Aufruf) die zugehörige Initialisierungsroutine gerufen (siehe Abschnitt Aufruf der Initialisierungsroutine zu einer benutzerdefinierten Anweisung). Ist keine Initialisierungsroutine definiert, wird die Anweisung @RUN mit der Meldung `EDT5469` abgewiesen.

Achtung

Das Format der Anweisung und die Schnittstelle, mit der die Routine gerufen wird, sind im Unicode-Modus anders als im Kompatibilitäts-Modus.

6 Produktion von Anwendungen der Unterprogramm-Schnittstelle

In diesem Abschnitt wird anhand von Regeln und Beispielen erläutert, wie im BS2000 Programme produziert werden können, die den EDT als Unterprogramm verwenden bzw. die vom EDT über die Anweisungen @USE oder @RUN als Anwenderroutinen aufgerufen werden sollen.

Dabei werden die Sprachen C und Assembler betrachtet. Für andere Sprachen bietet der EDT lediglich die ILCS-konforme Linkage an. Um das Nachladen oder Einbinden sowie die Initialisierung der benötigten Laufzeitroutinen muss sich der Anwender in diesen Fällen selbst kümmern.

6.1 Produktion von Hauptprogrammen in C

Es wird empfohlen, das Hauptprogramm mit `STDLIB=*DYNAMIC` zu binden (Standardwert) und den EDT mit ILCS-Flag `indicator.ilcs_environment = 1` (Standardwert) zu rufen. Die folgende Prozedur compiliert und bindet C-Hauptprogramme, die als `BEISPIEL1.C`, `BEISPIEL2.C`, ... in der Bibliothek `EDT.BEISPIELE` abgelegt sind.
Das erzeugte Programm, die Compiler-Listen und die Diagnose-Ausgabe werden ebenfalls in dieser Bibliothek abgelegt.

```
/SET-PROCEDURE-OPTIONS INPUT-FORMAT=FREE-RECORD-LENGTH,-
/    DATA-ESCAPE-CHAR=STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME=NR
/   DECLARE-PARAMETER NAME=LIB,INITIAL-VALUE=C'EDT.BEISPIELE'
/END-PARAMETER-DECLARATION
/START-CPLUS-COMPILER
//MODIFY-SOURCE-PROP LANGUAGE=*C
//MODIFY-RUNTIME-PROPERTIES PARAMETER-PROMPTING=*NO
//MODIFY-INCLUDE-LIBRARIES USER-INCLUDE-LIBRARY=-
//    $.SYSLIB.EDT.170                  ------------------------------ (1)
//MODIFY-DIAGNOSTIC-PROPERTIES OUTPUT=*LIBRARY-ELEMENT(-
//    LIB=&LIB,ELEM=*STD-ELEMENT()),-
//    MIN-MSG-WEIGHT=*NOTE
//MODIFY-LISTING-PROPERTIES OUTPUT=*LIBRARY-ELEMENT(-
```

```
//    LIB=&LIB,ELEM=*STD-ELEMENT()),SOURCE=*YES()
//COMPILE SOURCE=*LIB-ELEM(LIB=&LIB,ELEM=BEISPIEL&NR..C),-
//    MODULE-OUTPUT=*LIB-ELEM(LIB=&LIB,ELEM=BEISPIEL&NR)
//MOD-BIND-PROP INCLUDE=(*LIB(LIB=&LIB,ELEM=BEISPIEL&NR),-   ---------- (2)
//    *LIB(LIB=$.SYSLNK.EDT.170,ELEM=IEDTGLE)),-
//    RUNTIME-LANG=*C,STDLIB=*DYNAMIC ---------------------------------- (3)
//BIND OUTPUT=*LIB(LIB=&LIB,ELEM=BSP&NR.C)  -------------------------- (4)
//END
/EXIT-PROCEDUR
```

Erläuterungen

(1) Es wird angenommen, dass die SYSLIB.EDT.170 unter der Default-Userid installiert ist. Der C-Compiler benötigt diese Bibliothek, um die Header-Files der IEDTGLE-Schnittstelle zu finden.

(2) Das Benutzerprogramm wird mit dem Modul IEDTGLE aus der SYSLNK.EDT.170 zusammengebunden. IEDTGLE lokalisiert mittels IMON die EDT-Bibliotheken und lädt seinerseits den EDT dynamisch nach, bzw. konnektiert sich an das EDT-Subsystem.

(3) Der Adapter für das C-Laufzeitsystem wird vom Compiler automatisch eingebunden, wenn STDLIB=*DYNAMIC spezifiziert wird.

(4) Das erzeugte Programm kann anschließend mit /START-EXECUTABLE-PROGRAM (E=BSPxC,L=EDT.BEISPIELE) gestartet werden (x = 1,2,..).

Das ILCS-Flag hat nur dann eine Bedeutung, wenn vom EDT Anwenderroutinen geladen und aufgerufen werden sollen (siehe die folgenden Abschnitte).

6.2 Produktion von Anwenderroutinen in C

Eine C-Anwenderroutine, die nur aus einer Source produziert wird, kann ohne Bindeschritt als LLM in einer Bibliothek abgelegt werden. Der EDT gibt beim Laden der Anwenderroutine einen Resolve-Kontext an, der die benötigten Laufzeitmodule enthält.

Der genaue Mechanismus ist abhängig davon, ob der EDT von einem C-Hauptprogramm mit ILCS-Flag initialisiert wurde oder nicht:
Wenn dies der Fall ist, werden Anwenderroutinen in einen eigenen Ladekontext EDT#USER geladen und als Resolve-Kontext werden EDT#CRTS und LOCAL#DEFAULT (in dieser Reihenfolge) angegeben. Folglich werden Externverweise auf die C-Globals in der Anwenderroutine zuerst gegen die Entries aus EDT#CRTS (EDT-Laufzeitumgebung) und dann gegen die Entries aus LOCAL#DEFAULT (C-Hauptprogramm) abgesättigt.
Da eine (dynamisch nachgeladene) Anwenderroutine immer in der EDT-Laufzeitumgebung aufgerufen wird, werden der Anwenderroutine dadurch die richtigen C-Globals zur Verfügung gestellt, unabhängig davon, ob das C-Hauptprogramm seinerseits sichtbare Entries für die C-Globals hat.
Der eigene Ladekontext für die Anwenderroutine muss berücksichtigt werden, wenn aus der Anwenderroutine heraus weitere Module dynamisch mit BIND nachgeladen werden sollen.

Wenn das C-Hauptprogramm den EDT ohne ILCS-Flag initialisiert hat, wird die Anwenderroutine aus Kompatibilitätsgründen in den Ladekontext LOCAL#DEFAULT geladen und als Resolve-Kontext wird EDT#CRTS angegeben. In diesem Fall sollte das C-Hauptprogramm keine sichtbaren Entries auf die C-Globals haben, sonst majorisieren diese die entsprechenden Entries der EDT-Laufzeitumgebung und die Anwenderroutine kann nicht korrekt mit C-Bibliotheksfunktionen arbeiten.

Die folgende Prozedur compiliert und bindet C-Anwenderroutinen, die als ANWEND1.C, ANWEND2.C, ... in der Bibliothek EDT.BEISPIELE abgelegt sind.
Der erzeugte LLM, die Compiler-Listen und die Diagnose-Ausgabe werden ebenfalls in dieser Bibliothek abgelegt:

```
/SET-PROCEDURE-OPTIONS INPUT-FORMAT=FREE-RECORD-LENGTH,-
/    DATA-ESCAPE-CHAR=STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME=NR
/   DECLARE-PARAMETER NAME=LIB,INITIAL-VALUE=C'EDT.BEISPIELE'
/END-PARAMETER-DECLARATION
/START-CPLUS-COMPILER
//MODIFY-SOURCE-PROP LANGUAGE=*C
//MODIFY-RUNTIME-PROPERTIES PARAMETER-PROMPTING=*NO
//MODIFY-INCLUDE-LIBRARIES USER-INCLUDE-LIBRARY=-
//    $.SYSLIB.EDT.170                  ------------------------------ (1)
//MODIFY-DIAGNOSTIC-PROPERTIES OUTPUT=*LIBRARY-ELEMENT(-
//    LIB=&LIB,ELEM=*STD-ELEMENT()),-
```

```
//    MIN-MSG-WEIGHT=*NOTE
//MODIFY-MODULE-PROPERTIES LOWER-CASE-NAMES=*YES,-
//    SPECIAL-CHARACTERS=*KEEP          ------------------------------ (2)
//MODIFY-LISTING-PROPERTIES OUTPUT=*LIBRARY-ELEMENT(-
//    LIB=&LIB,ELEM=*STD-ELEMENT()),SOURCE=*YES()
//COMPILE SOURCE=*LIB-ELEM(LIB=&LIB,ELEM=ANWEND&NR..C),-
//    MODULE-OUTPUT=*LIB-ELEM(LIB=&LIB,ELEM=ANWEND&NR)
//END
/EXIT-PROCEDURE
```

Erläuterungen

(1) Es wird angenommen, dass die `SYSLIB.EDT.170` unter der Default-Userid installiert ist. Der C-Compiler benötigt diese Bibliothek, um die Header-Files der `IEDTGLE`-Schnittstelle zu finden.

(2) Die Entries in der Anwenderroutine sollen unter ihrem Originalnamen angesprochen werden können, daher müssen Kleinbuchstaben akzeptiert werden und Sonderzeichen (etwa "_") dürfen nicht verändert werden.

6.3 C-Hauptprogramm und Anwenderroutinen im gleichen Programm

Die Produktion eines Hauptprogramms, das auch Anwenderroutinen anbietet, unterscheidet sich nicht von dem in Abschnitt „Produktion von Hauptprogrammen in C“ auf Seite 103 beschriebenen Verfahren. Die dort angegebene Prozedur lässt sich unverändert verwenden. Man könnte lediglich zusätzlich die Anweisung

```
//MODIFY-MODULE-PROPERTIES LOWER-CASE-NAMES=*YES,-
//    SPECIAL-CHARACTERS=*KEEP
```

wie in Abschnitt „Produktion von Anwenderroutinen in C“ auf Seite 105 hinzunehmen, wenn man die Anwenderroutinen über den Originalnamen (Groß-/Kleinschreibung unterschieden) ansprechen möchte.

Wenn C-Hauptprogramm und Anwenderroutine im gleichen Programm liegen, muss der EDT *unbedingt* mit ILCS-Flag initialisiert werden.
Der EDT schaltet dann vor Aufruf der Anwenderroutine auf das Laufzeitsystem des C-Hauptprogramms um und ruft die Anwenderroutine mit diesem. Dieses Umschalten unterbleibt, wenn das ILCS-Flag nicht gesetzt ist, mit der Folge, dass die Anwenderroutine nicht die richtigen C-Globals benutzt und nicht korrekt mit C-Bibliotheksfunktionen arbeiten kann.

6.4 Produktion von Hauptprogrammen in Assembler

Es wird empfohlen, den Modul IEDTGLE explizit zum Hauptprogramm zu binden. Deshalb wird in der folgenden Prozedur an die Assemblierung noch ein Binderlauf angeschlossen. Im Allgemeinen wird ein reines Assembler-Programm den EDT ohne ILCS-Flag (EGLILCS) aufrufen. Das ist bei Generierung des Kontrollblocks IEDTGLCB der Standard. Die folgende Prozedur assembliert und bindet Assembler-Hauptprogramme, die als BEISPIEL1.ASS, BEISPIEL2.ASS, ... in der Bibliothek EDT.BEISPIELE abgelegt sind.
Das erzeugte Programm und die Compiler-Listen werden ebenfalls in dieser Bibliothek abgelegt.

```
/SET-PROCEDURE-OPTIONS INPUT-FORMAT=FREE-RECORD-LENGTH,-
/    DATA-ESCAPE-CHAR=STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME=NR
/   DECLARE-PARAMETER NAME=LIB,INITIAL-VALUE=C'EDT.BEISPIELE'
/END-PARAMETER-DECLARATION
/START-ASSEMBH
//COMPILE SOURCE=*LIBRARY-ELEMENT(-
//    LIBRARY=&LIB,ELEMENT=BEISPIEL&NR..ASS),-
//    MACRO-LIBRARY=($.SYSLIB.EDT.170,$.MACROLIB),-        ------------ (1)
//    COMPILER-ACTION=*MODULE-GENERATION(MODULE-FORMAT=*LLM),-
//    MODULE-LIBRARY=&LIB,-
//    LISTING=*PARAMETERS(OUTPUT=*LIBRARY-ELEMENT(-
//        LIBRARY=&LIB,ELEMENT=BEISPIEL&NR..LST))
//END
/START-BINDER
//START-LLM-CREATION INTERNAL-NAME=BEISPIEL&NR._ASS
//INCLUDE-MODULES MODULE-CONTAINER=*LIBRARY-ELEMENT(-
//    LIBRARY=&LIB,ELEMENT=BEISP&NR)                       ------------ (2)
//INCLUDE-MODULES MODULE-CONTAINER=*LIBRARY-ELEMENT(-
//    LIBRARY=$.SYSLNK.EDT.170,ELEMENT=IEDTGLE)            ------------ (3)
//SAVE-LLM MODULE-CONTAINER=*LIBRARY-ELEMENT(-
//    LIBRARY=&LIB,ELEMENT=BSP&NR.A)                       ------------ (4)
//END
/EXIT-PROCEDURE
```

Erläuterungen

(1) Es wird angenommen, dass die SYSLIB.EDT.170 unter der Default-Userid installiert ist. Der Assembler benötigt diese Bibliothek, um die Macros der IEDTGLE-Schnittstelle zu finden. Auf die Angabe der Standard-Makrolib kann i.d.R. verzichtet werden, evtl. sind weitere Makrobibliotheken anzugeben.

(2) Es wird angenommen, dass der Name der ersten CSECT des Assemblerprogramms BEISPx ist (BEISPIELx wäre im Standard-Assembler zu lang).

(3) Das Benutzerprogramm wird mit dem Modul IEDTGLE aus der SYSLNK.EDT.170 zusammengebunden. IEDTGLE lokalisiert mittels IMON die EDT-Bibliotheken und lädt seinerseits den EDT dynamisch nach, bzw. konnektiert sich an das EDT-Subsystem.

(4) Das erzeugte Programm kann anschließend mit /START-EXECUTABLE-PROGRAM (E=BSPxA,L=EDT.BEISPIELE) gestartet werden (x = 1,2,..).

6.5 Produktion von Anwenderroutinen in Assembler

Es wird empfohlen, den Modul IEDTGLE *nicht* mit einzubinden. Andernfalls müsste man die Entries dieses Moduls mittels BINDER verstecken, da es sonst beim Laden mehrerer Anwenderroutinen "duplicate Entries" geben würde.
Beim Nachladen werden die offenen Externals des Moduls dann entweder gegen einen IEDTGLE abgesättigt, der schon mit dem Hauptprogramm oder einer anderen Anwenderroutine geladen ist, oder es wird der mit dem Subsystem EDTCON vorgeladene IEDTGLE verwendet.
Die folgende Prozedur assembliert Anwenderroutinen, die als ANWEND1.ASS, ANWEND2.ASS, ... in der Bibliothek EDT.BEISPIELE abgelegt sind.
Das erzeugte Programm und die Compiler-Listen werden ebenfalls in dieser Bibliothek abgelegt.

```
/SET-PROCEDURE-OPTIONS INPUT-FORMAT=FREE-RECORD-LENGTH,-
/    DATA-ESCAPE-CHAR=STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME=NR
/   DECLARE-PARAMETER NAME=LIB,INITIAL-VALUE=C'EDT.BEISPIELE'
/END-PARAMETER-DECLARATION
/START-ASSEMBH
//COMPILE SOURCE=*LIBRARY-ELEMENT(-
//    LIBRARY=&LIB,ELEMENT=ANWEND&NR..ASS),-
//    MACRO-LIBRARY=($.SYSLIB.EDT.170,$.MACROLIB),-            -------- (1)
//    COMPILER-ACTION=*MODULE-GENERATION(MODULE-FORMAT=*LLM),-
//    MODULE-LIBRARY=&LIB,-
//    LISTING=*PARAMETERS(OUTPUT=*LIBRARY-ELEMENT(-
//        LIBRARY=&LIB,ELEMENT=ANWEND&NR..LST))
//END
/EXIT-PROCEDURE
```

Erläuterungen

(1) Es wird angenommen, dass die SYSLIB.EDT.170 unter der Default-Userid installiert ist. Der Assembler benötigt diese Bibliothek, um die Makros der IEDTGLE-Schnittstelle zu finden.
Auf die Angabe der Standard-Makrolib kann i.d.R. verzichtet werden. Eventuell sind weitere Makrobibliotheken anzugeben.

7 Beispiele

7.1 Beispiel 1 - C-Hauptprogramm

Das folgende Beispielprogramm benutzt nur die IEDTCMD-Schnittstelle.

```
/*********************************************************************/
/*                                                                   */
/* Beispiel 1                                                        */
/*                                                                   */
/* Dieses Beispiel verwendet ausschliesslich die IEDTCMD-            */
/* Schnittstelle zur Ausfuehrung von EDT-Anweisungen.                */
/*                                                                   */
/* Das Beispielprogramm fuehrt folgende Aktionen durch:              */
/*                                                                   */
/* 1) Einlesen eines Auswahlkriteriums (CCSN)                        */
/* 2) Ausgabe eines Inhaltsverzeichnisses in die Arbeitsdatei 0      */
/*    mit der @SHOW-Anweisung (Format 1).                            */
/* 3) Loeschen aller Zeilen, die nicht das Auswahlkriterium          */
/*    enthalten mit der @ON-Anweisung (Format 10).                   */
/* 4) Einstellen der Arbeitsdatei 0 und anschliessender Wechsel      */
/*    in den F-Modus-Dialog mit einer @SETF- und einer nach-         */
/*    folgenden @DIALOG-Anweisung.                                   */
/* 5) Der Anwender kann nun die ausgegebenen Zeilen editieren        */
/*    und schliesslich den EDT und damit auch dieses Beispiel-       */
/*    programm beenden.                                              */
/*                                                                   */
/*********************************************************************/

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
/* Include-Dateien der EDT-Unterprogrammschnittstelle */
#define EDT_V17                          /*--------------------------  (1) */
#include <iedtgle.h>                     /*--------------------------- (2) */

/* Anlegen und Initialisieren der fuer dieses Beispiel benoetigten */
/* Datenstrukturen der EDT-Unterprogrammschnittstelle.             */
```

```
static iedglcb glcb = IEDGLCB_INIT;
static iedupcb upcb = IEDUPCB_INIT;
static iedbuff *command = NULL;
static iedbuff *message1 = NULL;
static iedbuff *message2 = NULL;

/*********************************************************************/
/* Funktion: printrc                                                 */
/*                                                                   */
/* Aufgabe:                                                          */
/* Falls der EDT eine Fehlermeldung zurueckgegeben hat, wird die     */
/* Fehlermeldung durch diese Funktion ausgegeben.                    */
/*                                                                   */
/* Parameter: errmsg   (IN)  Zeiger auf die im Fehlerfall            */
/*                           zusaetzlich auszugebende Fehlermeldung  */
/*                                                                   */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
static void
printrc(char *errmsg)
{
    char message[81];

    if ((glcb.rc.structured_rc.mc.maincode != 0) &&
         (glcb.return_message.structured_msg.rmsgl > 0))
    {
        printf("%s\n",errmsg);   /* Uebergebene Fehlermeldung ausgeben */
        strncpy(message,(char*)glcb.return_message.structured_msg.rmsgf,
            glcb.return_message.structured_msg.rmsgl);
        message[glcb.return_message.structured_msg.rmsgl] = 0x00;

        printf("Meldungstext: %s\n",message);  /* EDT-Meldung ausgeben */
        exit(1);
    }
}
```

```
/*********************************************************************/
/* Funktion: fill_buff                                               */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion versorgt einen Satz variabler Laenge (DVS-Format) */
/* mit einem Inhalt sowie das Satzlaengenfeld.                       */
/*                                                                   */
/* Parameter: p:       (IN)  Zeiger auf eine Struktur vom Typ        */
/*                           iedbuff, die den zu versorgenden Satz   */
/*                           variabler Laenge enthaelt.              */
/*            textp:   (IN)  Zeiger auf einen String, der den ein-   */
/*                           zutragenden Text enthaelt. Die Laenge   */
/*                           des Strings legt implizit die Laenge    */
/*                           des Satzes fest (Laenge String + 4).    */
/*                                                                   */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
static void
fill_buff(iedbuff *p,char *textp)
{
    size_t l_text;                          /* Laenge des String */
    if ((l_text = strlen(textp)) > 2044)                 /*---------- (3) */
        l_text = 2044;     /* Laenge auf 2044 Zeichen begrenzen */
    strncpy((char *)p->text,textp,l_text);  /* Text eintragen */
    p->length = l_text + 4;                 /* Satzlaenge versorgen */
}

/*********************************************************************/
/* Funktion: edtcmd                                                  */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion traegt die uebergegebenen Strings in Saetze        */
/* variabler Laenge ein (DVS-Format) und ruft anschliessend die      */
/* CMD-Schnittstelle des EDT auf.                                    */
/*                                                                   */
/* Parameter: cmd:     (IN)  Zeiger auf einen String, der die        */
/*                           auszufuehrenden EDT-Anweisung(en)       */
/*                           enthaelt. Die Laenge des Strings legt   */
/*                           implizit die Laenge des Satze fest      */
/*                           (Laenge String + 4).                    */
/*            msg1:    (IN)  Zeiger auf einen String, der den ein-   */
/*                           zutragenden Text enthaelt. Die Laenge   */
/*                           des Strings legt implizit die Laenge    */
/*                           des Satzes fest (Laenge String + 4).    */
/*            msg2:    (IN)  Zeiger auf einen String, der den ein-   */
/*                           zutragenden Text enthaelt. Die Laenge   */
/*                           des Strings legt implizit die Laenge    */
/*                           des Satzes fest (Laenge String + 4).    */
```

```
/*                                                                  */
/* Rueckgabewert: keiner                                            */
/********************************************************************/
static void
edtcmd(char *cmd,char *msg1,char *msg2)
{
    fill_buff(command,cmd);
    fill_buff(message1,msg1);
    fill_buff(message2,msg2);
    IEDTCMD(&glcb,&upcb,command,message1,message2);
}

/********************************************************************/
/* Hauptprogramm                                                    */
/********************************************************************/
int
main(void)
{
    char input[81];     /* Eingabebereich */
    char ccsn[9];       /* eingegebener CCSN */
    char cmd[257];      /* Bereich zum Aufbau von EDT-Anweisungen */

    /* buffer bereitstellen */
    command = (iedbuff *)malloc(2048);
    message1 = (iedbuff *)malloc(2048);
    message2 = (iedbuff *)malloc(2048);

    printf("\nStart Beispiel1\n\n");

    /*  CCSN einlesen */

    printf("Bitte CCSN eingeben (UTFE,UTF16,EDFxxx): ");
    scanf("%s",input);
    if (strlen(input) < 1 || strlen(input) > 8)
    {
         printf("Eingabe zu kurz oder zu lang!");
         exit(1);
    }
    strupper(ccsn,input);      /* CCSN in Grossbuchst. konv. */

    /*  Inhaltsverzeichnis in die Arbeitsdatei 0 ausgeben */

    edtcmd("SHOW F=* TO 1 LONG","","");
    printrc("Fehler bei der @SHOW-Anweisung!");

    /*  Alle Zeilen loeschen, die nicht dem Suchkriterium (CCSN)  */
    /*  entsprechen und die verbliebenen Zeilen neu durchnummerieren */
```

```
    sprintf(cmd,"ON &:100-107: FIND NOT '%s' DELETE;RENUMBER",ccsn);
    edtcmd(cmd,"","");
    printrc("Fehler bei der @ON- oder der @RENUMBER-Anweisung!");

    /*  In die Arbeitsdatei 0 wechseln und in den */
    /*  F-Modus-Dialog umschalten                 */

    edtcmd("SETF(0);DIALOG","Beispiel 1 fuer die UP-Schnittstelle","");
    printrc("Fehler bei der @SETF- oder der @DIALOG-Anweisung!");

    edtcmd("HALT","","");
    printf("\nEnde Beispiel1\n\n");

    return 0;
}
```

Erläuterungen

(1) Durch das #define wird gesteuert, dass die V17-Variante der Schnittstellen generiert wird. Damit wird der EDT dann auch automatisch im Unicode-Modus gestartet.

(2) Mit EDT V17.0 genügt eine #include Anweisung auf iedtgle.h. Dieses Header-File zieht die weiteren benötigten Header nach.

(3) Die Begrenzung auf 2044 ist in diesem Programm durch den malloc für die Pufferbereiche bestimmt. Der EDT selbst verkraftet 32767 Byte.

Wenn die in Abschnitt Produktion von Hauptprogrammen in C erklärte Prozedur im BS2000 in einer Datei namens `CC.DO` und die Quelldatei als S-Element `BEISPIEL1.C` in der Bibliothek `EDT.BEISPIELE` abgelegt ist, kann das obige Programm mit

```
/CALL-PROC CC.DO,(1)
```

übersetzt und gebunden werden. Das erzeugte Programm ist anschließend mit

```
/START-EXECUTABLE-PROGRAM (E=BSP1C,L=EDT.BEISPIELE)
```

ausführbar. Bei Ablauf von `CC.DO` werden etwa folgende Ausgaben vom System bzw. vom Compiler erzeugt:

```
%  BLS0523 ELEMENT 'SDFCC', VERSION '031', TYPE 'L' FROM LIBRARY
':MARS:$TSOS.SYSLNK.CPP-RS.031' IN PROCESS
%  BLS0524 LLM 'SDFCC', VERSION '03.1A40' OF '2005-02-03 16:16:36' LOADED
%  BLS0551 COPYRIGHT (C) Fujitsu Siemens Computers GmbH 2005. ALL RIGHTS
RESERVED
%  CDR9992 : BEGIN C/C++(BS2000/OSD) VERSION 03.1A40
%  CDR9907 : NOTES: 0  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CDR9937 : MODULES GENERATED, CPU TIME USED = 2.4800 SEC
%  BND3102 SOME WEAK EXTERNS UNRESOLVED
```

```
%  BND1501 LLM FORMAT: '1'
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED EXTERNAL'
%  CDR9936 : END; SUMMARY: NOTES: 0  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CCM0998 CPU TIME USED: 3.4223 SECONDS
```

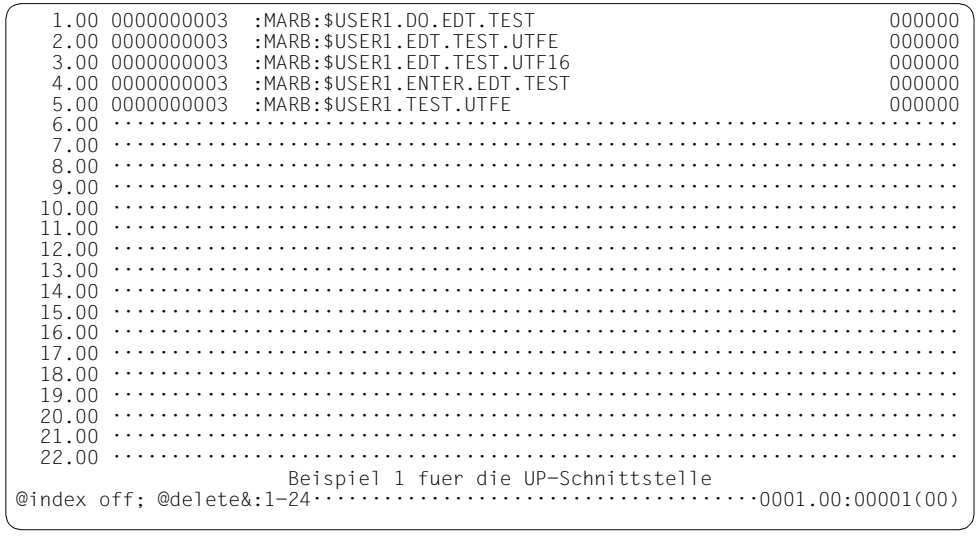
Nach Aufruf von `/START-EXECUTABLE-PROGRAM (E=BSP1C,L=EDT.BEISPIELE)` erscheinen etwa folgende Meldungen:

```
%  BLS0523 ELEMENT 'BSP1C', VERSION '@', TYPE 'L' FROM LIBRARY
':A:$USER.EDT.BEISPIELE' IN PROCESS
%  BLS0524 LLM '$LIB-ELEM$EDT$BEISPIELE$$BSP1C$$',VERSION' 'OF'2007-05-03 14
:42:12' LOADED

Start Beispiel1

Bitte CCSN eingeben (UTFE, UTF16, EDFxxx):
```

Gibt man hier `UTF` ein, d.h. wählt alle Dateien aus, die in einem Unicode-Zeichensatz codiert sind, wechselt das Programm anschließend in den EDT-Dialog und es erscheint folgender Bildschirm:



Durch Eingabe von @INDEX OFF; @DELETE&:1-24: beschränkt man die Ausgabe auf die relevante Information und erhält folgendes Bild:

```
DO.EDT.TEST                              0000000001 2006-10-05 NW SAM  NN UTFE
EDT.TEST.UTFE                            0000000001 2007-04-23 NW SAM  NN UTFE
EDT.TEST.UTF16                           0000000001 2007-04-23 NW SAM  NN UTF16
ENTER.EDT.TEST                           0000000001 2007-10-05 NW SAM  NN UTFE
TEST.UTFE                                0000000001 2007-03-22 NW SAM  NN UTF16
..............................................................................
..............................................................................



.............................................................0001.00:00001(00)
```

7.2 Beispiel 2 - C-Hauptprogramm

Dieses Beispiel nutzt die IEDTCMD-Schnittstelle und die IEDTGET-Schnittstelle, um Sätze einer Datei zu lesen und zu verarbeiten:

```
/*****************************************************************/
/*                                                               */
/* Beispiel 2                                                    */
/*                                                               */
/* Dieses Beispiel verwendet die IEDTCMD-Schnittstelle zur       */
/* Ausfuehrung von EDT-Anweisungen sowie die IEDGET-             */
/* Schnittstelle zum Lesen von Zeilen. Es werden aus einer       */
/* Datei, die Rechnungsdaten enthaelt, alle Datensaetze          */
/* ausgegeben, deren Rechnungsdatum mehr als eine fest           */
/* vorgegebene Anzahl von Tagen vor dem aktuellen Datum          */
/* liegen.                                                       */
/*                                                               */
/* Das Beispielprogramm fuehrt im Einzelnen folgende Aktionen    */
/* durch:                                                        */
/*                                                               */
/* 1) Aktuelles Datum ermitteln.                                 */
/* 2) Alle Zeilen der Datei mit den Rechnungsdaten mittels der   */
/*    @COPY-Anweisung (Format 1) in den Arbeitsbereich $0 einlesen. */
/* 3) In einer Schleife nacheinander alle Zeilen des Arbeits-    */
/*    bereichs $0 lesen und die Zeitdifferenz zwischen dem       */
/*    aktuellen Datum und dem Rechnungsdatum berechnen. Ist      */
/*    diese Zeitdifferenz groesser, als ein fest vorgegebenes    */
/*    Limit, wird von dieser Zeile Kundennummer, Rechnungsnummer */
/*    und Rechnungsbetrag ausgegeben.                            */
/* 4) Nach Beendigung der Schleife wird der EDT mit der Anweisung */
/*    @HALT und anschliessend dieses Beispielprogramm beendet.   */
/*                                                               */
/*****************************************************************/

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

/* Include-Dateien der EDT-Unterprogrammschnittstelle */
#define EDT_V17
#include <iedtgle.h>

/* Anlegen und Initialisieren der fuer dieses Beispiel benoetigten */
/* Datenstrukturen der EDT-Unterprogrammschnittstelle.             */
```

```
static iedglcb glcb = IEDGLCB_INIT;
static iedupcb upcb = IEDUPCB_INIT;
static iedamcb amcb = IEDAMCB_INIT;
static iedbuff *command = NULL;
static iedbuff *message1 = NULL;
static iedbuff *message2 = NULL;

/* Definition einer Ausgabezeile */

typedef struct line
{
    char kdnr[8];         /* Kundennummer */
    char free1[2];
    char renr[8];         /* Rechnungsnummer */
    char free2[2];
    char betr[10];        /* Rechnungsbetrag */
    char free3[2];
    char day[2];          /* Tag der Rechnungsstellung */
    char trenn1[1];
    char mon[2];          /* Monat der Rechnungsstellung */
    char trenn2[1];
    char year[4];         /* Jahr der Rechnungsstellung */
    char rest[215];
} LINE;

/*********************************************************************/
/* Funktion: printrc                                                 */
/*                                                                   */
/* Aufgabe:                                                          */
/* Falls der EDT eine Fehlermeldung zurueckgegeben hat, wird die     */
/* Fehlermeldung durch diese Funktion ausgegeben.                    */
/*                                                                   */
/* Parameter: errmsg   (IN)  Zeiger auf die im Fehlerfall            */
/*                           zusaetzlich auszugebende Fehlermeldung  */
/*                                                                   */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
static void
printrc(char *errmsg)
{
    char message[81];

    if (glcb.IEDGLCB_RC_MAINCODE != 0)
    {
        printf("%s: %08x\n",errmsg,
         glcb.IEDGLCB_RC_NBR);/* Uebergebene Fehlermeldung + code ausgeben */
        if (glcb.return_message.structured_msg.rmsgl > 0)
        {
```

```
            strncpy(message,(char*)glcb.return_message.structured_msg.rmsgf,
                glcb.return_message.structured_msg.rmsgl);
            message[glcb.return_message.structured_msg.rmsgl] = 0x00;

            printf("Meldungstext: %s\n",message);  /* EDT-Meldung ausgeben */
        }
        exit(1);
    }
}

/*********************************************************************/
/* Funktion: fill_buff                                               */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion versorgt einen Satz variabler Laenge (DVS-Format) */
/* mit einem Inhalt sowie das Satzlaengenfeld.                       */
/*                                                                   */
/* Parameter: p:       (IN)  Zeiger auf eine Struktur vom Typ       */
/*                           iedbuff, die den zu versorgenden Satz  */
/*                           variabler Laenge enthaelt.             */
/*            textp:   (IN)  Zeiger auf einen String, der den ein-  */
/*                           zutragenden Text enthaelt. Die Laenge  */
/*                           des Strings legt implizit die Laenge   */
/*                           des Satzes fest (Laenge String + 4).   */
/*                                                                   */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
static void
fill_buff(iedbuff *p,char *textp)
{
    size_t l_text;                          /* Laenge des String */
    if ((l_text = strlen(textp)) > 2044)
        l_text = 2044;      /* Laenge auf 2044 Zeichen begrenzen */
    strncpy((char *)p->text,textp,l_text);  /* Text eintragen */
    p->length = l_text + 4;                 /* Satzlaenge versorgen */
}

/*********************************************************************/
/* Funktion: edtcmd                                                  */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion traegt die uebergegebenen Strings in Saetze       */
/* variabler Laenge ein (DVS-Format) und ruft anschliessend die     */
/* CMD-Schnittstelle des EDT auf.                                    */
/*                                                                   */
/* Parameter: cmd:     (IN)  Zeiger auf einen String, der die       */
/*                           auszufuehrenden EDT-Anweisung(en)      */
/*                           enthaelt. Die Laenge des Strings legt  */
```

```
/*                          implizit die Laenge des Satze fest     */
/*                          (Laenge String + 4).                   */
/*            msg1:    (IN)  Zeiger auf einen String, der den ein-  */
/*                          zutragenden Text enthaelt. Die Laenge  */
/*                          des Strings legt implizit die Laenge   */
/*                          des Satzes fest (Laenge String + 4).   */
/*            msg2:    (IN)  Zeiger auf einen String, der den ein-  */
/*                          zutragenden Text enthaelt. Die Laenge  */
/*                          des Strings legt implizit die Laenge   */
/*                          des Satzes fest (Laenge String + 4).   */
/*                                                                 */
/* Rueckgabewert: keiner                                           */
/*******************************************************************/
static void
edtcmd(char *cmd,char *msg1,char *msg2)
{
    fill_buff(command,cmd);
    fill_buff(message1,msg1);
    fill_buff(message2,msg2);
    IEDTCMD(&glcb,&upcb,command,message1,message2);
}

/*******************************************************************/
/* Funktion: edtget                                                */
/*                                                                 */
/* Aufgabe:                                                        */
/* Diese Funktion liesst mit der Funktion GET der Unterprogramm-   */
/* Schnittstelle eine Zeile aus der Arbeitsdatei $0. Es wird immer */
/* relativ zum Satz mit Zeilennummer 0 gelesen.                    */
/*                                                                 */
/* Parameter: rec:     (IN)  Zeiger auf einen Datenbereich, in dem */
/*                           die von der Funktion GET gelesene     */
/*                           Zeile abgelegt wird.                  */
/*            disp:    (IN)  Gibt an, die wievielte Zeile ab der   */
/*                           Zeilennummer 0 gelesen werden soll.   */
/* Rueckgabewert: keiner                                           */
/*******************************************************************/
static void
edtget(char *rec,int disp)
{
    char localfile[9] = "$0      ";
    iedbyte key1[8] = {0,0,0,0,0,0,0,0};
    iedbyte key[8] = {0,0,0,0,0,0,0,0};

    /* Kontrollblock IEDAMCB versorgen */
    IEDAMCB_SET_NO_MARKS(amcb);
    amcb.length_key_outbuffer = 8;
    amcb.length_rec_outbuffer = 256;
```

```
    amcb.length_key1 = 8;              /* -------------------------- (1) */
    amcb.displacement = disp;  /* lese <disp> Sätze nach Nr. 0 */
    strncpy((char *)amcb.filename,localfile,8); /* Arbeitsdatei $0 */
    IEDTGET(&glcb,&amcb,key1,key,rec);
}


/*********************************************************************/
/* Hauptprogramm                                                     */
/*********************************************************************/

int
main(void)
{
    char filename[] = "edt.rechnung2";
    char cmd[257];
    LINE zeile;
    int  disp;
    int  days = 20;
    int  time_diff;
    time_t zeit1;
    time_t zeit2;
    struct tm t;

    printf("\nStart Beispiel2\n\n");

    /* buffer bereitstellen */
    command = (iedbuff *)malloc(2048);
    message1 = (iedbuff *)malloc(2048);
    message2 = (iedbuff *)malloc(2048);

    /* Heutiges Datum ermitteln */
    zeit1 = time((time_t *) 0);

    /* Fuer die Umwandlung des Datums in der Zeile */
    /* wird die Uhrzeit auf 0 Uhr gesetzt          */
    t.tm_sec = 0;
    t.tm_min = 0;
    t.tm_hour = 0;

    /* Die zu bearbeitende Datei mit der   */
    /* Anweisung @COPY (Format 1) einlesen */
    sprintf(cmd,"COPY FILE=%s",filename);
    edtcmd(cmd,"","");
    printrc("Fehler bei der @COPY-Anweisung!");

    /* In der folgenden Schleife werden alle Zeilen */
    /* der Arbeitsdatei $0 bearbeitet.              */
```

```
    for (disp = 1; disp < 99999999; disp++)   /*-------------------- (2) */
    {
        /* Naechste Zeile lesen */
        edtget((char *)&zeile,disp);
        printrc("Fehler bei der Funktion GET!");

        /* Wenn jenseits der letzten Zeilennummer gelesen wird */
        /* gibt IEDTGET "last record" zurück */
        if (glcb.IEDGLCB_RC_SUBCODE1 == IEDGLCBlast_record)
            break; /* Schleife verlassen */

        t.tm_mday = atoi(zeile.day);          /* Datum aus der  */
        t.tm_mon = atoi(zeile.mon) - 1;       /* eingelesenen   */
        t.tm_year = atoi(zeile.year) - 1900;  /* Zeile holen    */
        zeit2 = mktime(&t);                   /* Datum in einen */
                                              /* Zeitwert konv. */

        /* Zeitdifferenz in Tagen bestimmen */
        time_diff = difftime(zeit1,zeit2)/86400;

        /* Falls die vorgegebene Zeitspanne vorbei ist, fuer die */
        /* aktuelle Zeile Kundennummer, Rechnungsnummer und      */
        /* Rechnungsbetrag ausgeben.                             */
        if (time_diff > days)
        {
            zeile.free1[0] = '\0';   /* ---------------------------- (3) */
            zeile.free2[0] = '\0';
            zeile.free3[0] = '\0';

            printf("Kdnr.: %s, Rechn.nr.: %s, Betrag: %s Euro\n",
                 zeile.kdnr,zeile.renr,zeile.betr);
        }
    }

    /* EDT und Programm beenden */
    edtcmd("HALT","","");
    printrc("Fehler bei der @HALT-Anweisung!");

    printf("\n\nEnde Beispiel2\n\n");
    return 0;
}
```

Erläuterungen

(1) Bei der IEDTGET-Funktion müssen nur length_key1 und length_key_outbuffer versorgt sein.

(2) Die hier implementierte Methode zum sequenziellen Lesen der Arbeitsdatei ist nicht sonderlich effektiv, da immer wieder von Beginn an gelesen wird. Man überlege sich, wie man unter Verwendung des in key zurück gelieferten Schlüssels mit konstantem amcb.displacement = +1 einen günstigeren Algorithmus implementieren kann.

(3) Hier wird das Endezeichen für C-Strings eingesetzt, um eine direkte Ausgabe ohne Umwandlung zu ermöglichen.

Wenn die in Abschnitt „Produktion von Hauptprogrammen in C“ auf Seite 103 erklärte Prozedur im BS2000 in einer Datei namens CC.DO und die Quelldatei als S-Element BEISPIEL2.C in der Bibliothek EDT.BEISPIELE abgelegt ist, kann das obige Programm mit

```
/CALL-PROC CC.DO,(2)
```

übersetzt und gebunden werden. Das erzeugte Programm ist anschließend mit

```
/START-EXECUTABLE-PROGRAM (E=BSP2C,L=EDT.BEISPIELE)
```

ausführbar. Bei Ablauf von CC.DO werden etwa folgende Ausgaben vom System bzw. vom Compiler erzeugt:

```
%  BLS0524 LLM 'SDFCC', VERSION '03.1A40' OF '2005-02-03 16:16:36' LOADED
%  BLS0551 COPYRIGHT (C) Fujitsu Siemens Computers GmbH 2005. ALL RIGHTS
RESERVED
%  CDR9992 : BEGIN C/C++(BS2000/OSD) VERSION 03.1A40
%  CDR9907 : NOTES: 0  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CDR9937 : MODULES GENERATED, CPU TIME USED = 2.6000 SEC
%  BND3102 SOME WEAK EXTERNS UNRESOLVED
%  BND1501 LLM FORMAT: '1'
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED EXTERNAL'
%  CDR9936 : END; SUMMARY: NOTES: 0  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CCM0998 CPU TIME USED: 3.6403 SECONDS
```

Die von diesem Beispielprogramm zu bearbeitende Datei EDT.RECHNUNG2 sehe folgendermaßen aus:

```
00347563  00028654     1378.89  21.05.2007
00345781  00027349    21500.00  07.04.2007
00375863  00028937      248.23  19.05.2007
00242365  00012358     4577.54  23.03.2007
00416467  00046687     6776.31  10.05.2007
00576373  00015463      578.00  19.04.2007
00785214  00053417    65465.00  13.12.2006
00265432  00065743     6534.67  16.05.2007
```

```
00546315  00035476      656.34  29.05.2007
00675436  00015334     7878.45  04.05.2007
00353466  00087227      654.24  11.11.2006
00534267  00067854    52346.00  15.05.2007
00243535  00078921     4532.54  22.04.2007
00783432  00063223      548.19  17.05.2007
00623556  00054342     5346.32  17.03.2007
00234354  00065233     4534.65  08.05.2007
```

Nach Aufruf von /START-EXECUTABLE-PROGRAM (E=BSP2C,L=EDT.BEISPIELE) wird dann (am 4.5.2007) in etwa folgende Ausgabe zu sehen sein:

```
%  BLS0523 ELEMENT 'BSP2C', VERSION '@', TYPE 'L' FROM LIBRARY ':A:$USER.EDT
.BEISPIELE' IN PROCESS
%  BLS0524 LLM '$LIB-ELEM$EDT$BEISPIELE$$BSP2C$$', VERSION ' ' OF '2007-05-04
12
:22:24' LOADED

Start Beispiel2

Kdnr.: 00345781, Rechn.nr.: 00027349, Betrag:   21500.00 Euro
Kdnr.: 00242365, Rechn.nr.: 00012358, Betrag:    4577.54 Euro
Kdnr.: 00785214, Rechn.nr.: 00053417, Betrag:   65465.00 Euro
Kdnr.: 00353466, Rechn.nr.: 00087227, Betrag:     654.24 Euro
Kdnr.: 00623556, Rechn.nr.: 00054342, Betrag:    5346.32 Euro


Ende Beispiel  2

%  CCM0998 CPU TIME USED: 0.3276 SECONDS
```

7.3 Beispiel 3 - C-Anwenderroutine

Die folgende Anwenderroutine dient dazu, numerische Werte innerhalb eines Spaltenbereichs der aktuellen Arbeitsdatei zu addieren und das Ergebnis in die letzte Zeile einzufügen.
Es werden die EDT-Funktionen `IEDTEXE`, `IEDTGET`, `IEDTPUT` und `IEDTPTM` verwendet.

```
/*****************************************************************/
/*                                                               */
/* Beispiel 3                                                    */
/*                                                               */
/* Dies ist ein Beispiel für eine Anwenderroutine. Sie wird mit  */
/* USE COM='*',ENTRY=*,MODLIB=EDT.BEISPIELE vereinbart.          */
/* Anschließend wird über *sum <col1>-<col2> die Funktion SUM    */
/* gerufen, wobei nur noch das Argument <col1>-<col2> übergeben  */
/* wird. Die Funktion SUM liest alle Zeilen der aktuellen Arbeits- */
/* datei und extrahiert die im Spaltenbereich <col1>-<col2> ent- */
/* haltenen Zahlenwerte. Zeilen mit ungültigen Werten werden mit */
/* Markierung 14 versehen (werden dann überschreibbar gestellt). */
/* Die letzte gelesene Zeile nimmt den Summenwert auf.           */
/*                                                               */
/* Das Beispielprogramm fuehrt im Einzelnen folgende Aktionen    */
/* durch:                                                        */
/*                                                               */
/* 1) Den gewünschten Spaltenbereich aus den Aufrufargumenten    */
/*    ermitteln.                                                 */
/* 2) In einer Schleife nacheinander alle Zeilen des aktuellen   */
/*    Arbeitsbereichs lesen und die im gewünschten Spaltenbereich */
/*    enthaltene Zahl in eine Gleitpuktzahl umwandeln. Wenn dies */
/*    gelingt, Gleitpunktzahl zum Summenwert addieren. Wenn dies */
/*    nicht gelingt, Zeile mit Markierung 14 kennzeichnen (sie wird */
/*    dann überschreibbar gestellt).                             */
/* 4) Nach Beendigung der Schleife wird der Summenwert in die    */
/*    letzte Zeile eingefügt und zum Aufrufer zurückgekehrt.     */
/*                                                               */
/*****************************************************************/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

/* Include-Dateien der EDT-Unterprogrammschnittstelle */
#define EDT_V17
#include <iedtgle.h>
```

```
/* Alternative zur fill_buff Funktion: Macro zum Füllen des Puffers */
                                       /* ---------------------------- (1) */
#define IEDBUFF_FILL(buf,s) \
    (buf)->length = (strlen(s) + 4); \
    strncpy((char *)(buf)->text,s,(size_t)((buf)->length - 4))

/*********************************************************************/
/* Funktion: edtget                                                  */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion liest mit der Funktion IEDTGET der Unterprogramm-  */
/* Schnittstelle eine Zeile aus der aktuellen Arbeitsdatei. Es wird  */
/* immer relativ zum Satz mit Zeilennummer 0 gelesen.                */
/*                                                                   */
/* Parameter: glcb:    (IN)  Zeiger auf den von EDT mitgelieferten   */
/*                           glcb.                                   */
/*            rec:     (IN)  Zeiger auf einen Datenbereich, in dem   */
/*                           die von der Funktion GET gelesene       */
/*                           Zeile abgelegt wird.                    */
/*            disp:    (IN)  Gibt an, die wievielte Zeile ab der     */
/*                           Zeilennummer 0 gelesen werden soll.     */
/*            key      (OUT) Zeiger auf ein Feld, in dem der gele-   */
/*                           sene Schlüssel hinterlegt wird          */
/*********************************************************************/
static void
edtget(iedglcb *glcb,char *rec,int disp,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;
    iedbyte key1[8] = {0,0,0,0,0,0,0,0};

    /* Kontrollblock IEDAMCB versorgen */
    IEDAMCB_SET_NO_MARKS(amcb);
    amcb.length_key_outbuffer = 8;
    amcb.length_rec_outbuffer = 256;
    amcb.length_key1 = 8;
    amcb.displacement = disp;  /* lese <disp> Sätze nach Nr. 0 */
    /* Nimm aktuelle Arbeitsdatei (aus glcb) */
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
    IEDTGET(glcb,&amcb,key1,key,rec);
    rec[amcb.length_rec] = 0;  /* set C-string end */
}

/*********************************************************************/
/* Funktion: edtput                                                  */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion schreibt mit der IEDTPUT-Funktion einen Satz an    */
/* die durch den Schlüssel key bezeichnete Stelle.                   */
```

```
/*                                                                  */
/* Parameter: glcb:    (IN)  Zeiger auf den von EDT mitgelieferten  */
/*                           glcb.                                  */
/*            rec:     (IN)  Zeiger auf einen Datenbereich, in dem  */
/*                           der zu schreibende Satz übergeben wird.*/
/*            key      (IN)  Zeiger auf ein Feld, in dem der zu     */
/*                           (über-)schreibende Schlüssel steht.    */
/* Rueckgabewert: keiner                                            */
/********************************************************************/
static void
edtput(iedglcb *glcb,char *rec,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;

    amcb.length_key = 8;
    amcb.marks.mark_field = 0;
    amcb.length_rec = strlen(rec);
    /* Nimm aktuelle Arbeitsdatei (aus glcb) */
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
    IEDTPUT(glcb,&amcb,key,(unsigned char *)rec);
}

/********************************************************************/
/* Funktion: edtptm                                                 */
/*                                                                  */
/* Aufgabe:                                                         */
/* Diese Funktion schreibt mit der IEDTPTM-Funktion eine Markierung */
/* an die durch den Schlüssel key bezeichnete Stelle.               */
/*                                                                  */
/* Parameter: glcb:    (IN)  Zeiger auf den von EDT mitgelieferten  */
/*                           glcb.                                  */
/*            mark:    (IN)  Wert der zu schreibenden Markierung.   */
/*            key      (IN)  Zeiger auf ein Feld, in dem der zu     */
/*                           markierende Schlüssel steht.           */
/* Rueckgabewert: keiner                                            */
/********************************************************************/
static void
edtptm(iedglcb *glcb,int mark,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;

    amcb.marks.mark_field = 0;
    amcb.length_key = 8;
    if (mark != 0)                /* ---------------------------------- (2) */
        amcb.marks.mark_field = (1 << mark);
    /* Nimm aktuelle Arbeitsdatei (aus glcb) */
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
```

```
    IEDTPTM(glcb,&amcb,key);
}

/*********************************************************************/
/* Funktion: SUM                                                     */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion liest mit der Funktion edtget die gesamte          */
/* Arbeitsdatei, extrahiert für jede Zeile den Spaltenbereich,       */
/* der als Argument mitgeliefert wurde, und versucht, den dort       */
/* enthaltenen Text in eine Gleitkommazahl umzuwandeln.              */
/* Die ermittelten Zahlen werden addiert und der Summenwert wird     */
/* in die letzte Zeile ausgegeben (überschreibt dort den gegebenen   */
/* Spaltenbereich).                                                  */
/* Zeilen, die an der gegebenen Stelle keine gültige Gleitpunkt-     */
/* zahl enthalten, werden mittels Markierung 14 überschreibbar       */
/* gestellt.                                                         */
/*                                                                   */
/* Parameter: glcb:    (IN)  Zeiger auf einen glcb, der für Aufrufe */
/*                           von IEDTGET oder IEDTPUT verwendet      */
/*                           wird.                                   */
/*            cmd:     (IN)  Argumente, mit denen sum aufgerufen     */
/*                           wurde.                                  */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
void
SUM(iedglcb *glcb,iedbuff *cmd)         /* --------------------------- (3) */
{
    char command[80];
    char line[256];
    char number[64];
    int len = cmd->length;
    size_t col1 = 0;
    size_t col2 = 0;
    iedbyte key[8];
    float fnum;
    float sum;
    int disp;

    /* Ermittle Spaltenbereich aus dem Aufrufargument */
    cmd->text[len] = '\0'; /* setze Endezeichen für C-String */
    if (sscanf((char *)cmd->text," %d - %d ",&col1,&col2) < 2)
    {
        glcb->IEDGLCB_RC_MAINCODE = IEDGLCBcmd_unrec_user_error;
        glcb->IEDGLCB_RC_SUBCODE1 = IEDGLCBparameter_error;
        return;                       /*------------------------------ (4) */
    }
    /* Anweisung @PAR PROT=ON, damit Markierung 14 wirkt */
```

```
    IEDBUFF_FILL((iedbuff *)command,"@PAR PROT=ON");
    IEDTEXE(glcb,(iedbuff *)command);
    sum = 0.0;
    /* Lese alle Zeilen der aktuellen Arbeitsdatei */
    for (disp = 1; disp < 99999999; disp++)  /*----------------------- (5) */
    {
        /* Naechste Zeile lesen */
        edtget(glcb,(char *)&line,disp,key);
        if (glcb->IEDGLCB_RC_MAINCODE != 0)
            return; /* MAINCODE unverändert durchreichen */

        /* Wenn jenseits der letzten Zeilennummer gelesen wird */
        /* gibt IEDTGET "last record" zurück */
        if (glcb->IEDGLCB_RC_SUBCODE1 == IEDGLCBlast_record)
            break; /* Schleife verlassen */

        /* extrahiere Zahlenwert */
        strncpy(number,&line[col1 - 1],col2 - col1 + 1);
        number[col2 - col1 + 1] = '\0';
        if (sscanf(number," %f ",&fnum) >= 1)
        {
            sum += fnum;
            edtptm(glcb,0,key); /* Setze evtl. Markierung zurück */
        }
        else /* keine gültige Gleitpunktzahl im Bereich */
        {
            /* Setze Markierung 14 - Zeile wird überschreibbar gestellt */
            edtptm(glcb,14,key);
        }
    }
    /* Setze Summenwert in die letzte Zeile ein und schreibe sie zurück */
    sprintf(number,"%6.2f      ",sum);
    strncpy(&line[col1 - 1],number,col2 - col1 + 1);
    edtput(glcb,(char *)line,key);
}

/*********************************************************************/
/* Funktion: SUM@I                                                   */
/*                                                                   */
/* Aufgabe:                                                          */
/* Dies ist die Initialisierungsroutine zu SUM.                      */
/*                                                                   */
/* Parameter: glcb:    (IN)  Zeiger auf einen glcb in dem u.a. der   */
/*                           beim Aufruf von sum zu verwendende      */
/*                           Zeichensatz festgelegt werden kann.     */
/*                                                                   */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
```

```
void SUM@I(iedglcb *glcb)                /* -------------------------- (6) */
{
    glcb->indicator.compatible_format = 1;
    memcpy(glcb->ccsn,"EDF04F    ",8); /* sum erwartet EDF04F */
    glcb->rc.rc_nbr = 0;
}
```

Erläuterungen

(1) Als Alternative zur Funktion `fill_buff` aus den Beispielen 1 und 2 wird hier ein mit `#define` vereinbartes Makro verwendet, um einen Anweisungspuffer zu versorgen.

(2) Die Funktion `edtptm` setzt jeweils nur eine Markierung. Zum Löschen einer Markierung muss sie mit dem Wert `0` aufgerufen werden.

(3) Der Funktionsname wird in Großbuchstaben definiert, da die Benutzeranwesung mit @USE COM='*',ENTRY=*,... vereinbart werden und mit `*sum c1 - c2` aufgerufen werden soll. In diesem Fall setzt der EDT den ersten Teil der Benutzeranweisung (den Anweisungsnamen) in Großbuchstaben um (siehe Abschnitt „Aufruf einer benutzerdefinierten Anweisung“ auf Seite 91).

(4) Wenn sich das Aufrufargument nicht in zwei Ganzzahlen (Anfangs- und Endspalte) umwandeln lässt, wird die Anwenderroutine mit Returncode verlassen. Es erscheint dann die Meldung `EDT5410`. Hier sollte man natürlich noch weitere Überprüfungen (z.B. `col1 < col2`) einfügen und entsprechende Meldungstexte aufbauen.
Im Interesse der Übersichtlichkeit wurde im Beispiel darauf verzichtet.

(5) Die hier implementierte Methode zum sequenziellen Lesen der Arbeitsdatei ist nicht sonderlich effektiv, da immer wieder von Beginn an gelesen wird. Man überlege sich, wie man unter Verwendung des in `key` zurück gelieferten Schlüssels mit konstantem `amcb.displacement = +1` einen günstigeren Algorithmus implementieren kann.

(6) Es wird eine Initialisierungsroutine eingerichtet, damit der Aufruf der Anwenderroutine über die V17-Schnittstelle und im Zeichensatz `EDF04F` erfolgt.

Wenn die in Abschnitt „Produktion von Anwenderroutinen in C“ auf Seite 105 erklärte Prozedur im BS2000 in einer Datei namens `CCMOD.DO` und die Quelldatei als S-Element `ANWEND1.C` in der Bibliothek `EDT.BEISPIELE` abgelegt ist, kann das obige Programm mit

```
/CALL-PROC CCMOD.DO,(1)
```

übersetzt und das `LLM ANWEND1` in der Bibliothek `EDT.BEISPIELE` abgelegt werden.

Wenn man die Benutzeranweisung mit

@USE COMMAND='*',ENTRY=*,MODLIB=EDT.BEISPIELE

vereinbart, kann man die Anweisung `*SUM` etwa benutzen, um Preise in einer Einkaufsliste addieren zu lassen:

```
   1.00 Butter           1.50 €<·····································
   2.00 Quark            1.20 €<·····································
   3.00 Milch            1.10 €<·····································
   4.00 Käse             3.79 €<·····································
   5.00 Gummibärchen     3.30 €<·····································
   6.00 Schlagsahne      2.50 €<·····································
   7.00 Crème fraiche    1.80 €<·····································
   8.00 Sauerrahm        0.90 €<·····································
   9.00 Karamelbonbons   2.20 €<·····································
  10.00 Puddingpulver    0.70 €<·····································
  11.00 Pumpernickel     2.30 €<·····································
  12.00 SUMME           00.00 €<·····································
  13.00 ·····································································
  14.00 ·····································································
  15.00 ·····································································
  16.00 ·····································································
  17.00 ·····································································
  18.00 ·····································································
  19.00 ·····································································
  20.00 ·····································································
  21.00 ·····································································
  22.00 ·····································································
  23.00·····································································
*sum 16-21·····················································0001.00:00001(00)
```

Nach Eingabe von `*sum 16-21` wird folgender Bildschirm angezeigt:

```
   1.00 Butter           1.50 €<·····································
   2.00 Quark            1.20 €<·····································
   3.00 Milch            1.10 €<·····································
   4.00 Käse             3.79 €<·····································
   5.00 Gummibärchen     3.30 €<·····································
   6.00 Schlagsahne      2.50 €<·····································
   7.00 Crème fraiche    1.80 €<·····································
   8.00 Sauerrahm        0.90 €<·····································
   9.00 Karamelbonbons   2.20 €<·····································
  10.00 Puddingpulver    0.70 €<·····································
  11.00 Pumpernickel     2.30 €<·····································
  12.00 SUMME           21.29 €<·····································
  13.00 ·····································································
  14.00 ·····································································
  15.00 ·····································································
  16.00 ·····································································
  17.00 ·····································································
  18.00 ·····································································
  19.00 ·····································································
  20.00 ·····································································
  21.00 ·····································································
  22.00 ·····································································
  23.00·····································································
·····························································0001.00:00001(00)
```

7.4 Beispiel 4 - C-Hauptprogramm und Anwenderroutine in einer Source

Dieses Beispiel zeigt, wie man ein C-Hauptprogramm und eine C-Anwenderroutine in der gleichen Source compilieren kann.
Da der EDT bei der @USE-Anweisung zunächst den angegebenen Entry mit `VSVI` sucht, findet er den Entry im Hauptprogramm und verwendet diesen.
Ein Nachladen findet dann nicht statt.
Im Beispiel wird die @USE-Anweisung auch gleich im Hauptprogramm gegeben, so dass der Anwender die Benutzeranweisung `*rot` (o.ä.) gleich zur Verfügung hat.
Da die Anwenderroutine keinen speziellen Anweisungsstring erwartet, kann Beispiel 4 auch als Beispiel für eine Anwenderroutine dienen, die mit @RUN ENTRY=ROT13 aufgerufen werden kann.

```
/*********************************************************************/
/*                                                                   */
/* Beispiel 4                                                        */
/*                                                                   */
/* Dieses Beispiel zeigt die Kombination eines C-Hauptprogramms      */
/* mit einer Anwenderroutine. Es gestattet die Übergabe eines        */
/* Dateinamens als Argument. Diese Datei wird eingelesen, dann wird  */
/* in den Bildschirmdialog gewechselt. Die Anwenderroutine im        */
/* gleichen Programm ver- und etschlüsselt alle Sätze der aktuellen  */
/* Arbeitsdatei nach dem (primitiven) ROT13 Verfahren.               */
/*                                                                   */
/* Das Hauptprogramm fuehrt folgende Aktionen durch:                 */
/*                                                                   */
/* 1) Ermitteln des Dateinamens aus dem Aufrufargument.              */
/* 2) Einlesen der Datei mittels @OPEN-Anweisung über IEDTCMD        */
/* 3) Benutzeranweisung "*ROT13" mit @USE einrichten                 */
/* 3) Verzweigen in den Bildschirmdialog mit @DIALOG                 */
/*                                                                   */
/* Die Anwenderroutine ROT13 fuehrt folgende Aktionen durch:         */
/*                                                                   */
/* 1) Lesen der aller Sätze der aktuellen Arbeitsdatei in Schleife.  */
/* 2) Verschlüsseln jedes Satzes mit ROT13.                          */
/* 3) Rückschreiben des Satzes in die Arbeitsdatei mit IEDTPUT       */
/*                                                                   */
/*********************************************************************/

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
```

```
/* Include-Dateien der EDT-Unterprogrammschnittstelle */
#define EDT_V17
#include <iedtgle.h>

/* Anlegen und Initialisieren der fuer dieses Beispiel benoetigten */
/* Datenstrukturen der EDT-Unterprogrammschnittstelle.             */

static iedglcb glcb = IEDGLCB_INIT;
static iedupcb upcb = IEDUPCB_INIT;
static iedbuff *command = NULL;
static iedbuff *message1 = NULL;
static iedbuff *message2 = NULL;

/*********************************************************************/
/* Funktion: printrc                                                 */
/*                                                                   */
/* Aufgabe:                                                          */
/* Falls der EDT eine Fehlermeldung zurueckgegeben hat, wird die     */
/* Fehlermeldung durch diese Funktion ausgegeben.                    */
/*                                                                   */
/* Parameter: errmsg   (IN)  Zeiger auf die im Fehlerfall            */
/*                           zusaetzlich auszugebende Fehlermeldung */
/*                                                                   */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
static void
printrc(char *errmsg)
{
    char message[81];

    if ((glcb.rc.structured_rc.mc.maincode != 0) &&
         (glcb.return_message.structured_msg.rmsgl > 0))
    {
        printf("%s\n",errmsg);   /* Uebergebene Fehlermeldung ausgeben */

        strncpy(message,(char*)glcb.return_message.structured_msg.rmsgf,
            glcb.return_message.structured_msg.rmsgl);
        message[glcb.return_message.structured_msg.rmsgl] = 0x00;

        printf("Meldungstext: %s\n",message);  /* EDT-Meldung ausgeben */

        exit(1);
    }
}
```

```
/*********************************************************************/
/* Funktion: fill_buff                                               */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion versorgt einen Satz variabler Laenge (DVS-Format) */
/* mit einem Inhalt sowie das Satzlaengenfeld.                       */
/*                                                                   */
/* Parameter: p:       (IN)  Zeiger auf eine Struktur vom Typ        */
/*                           iedbuff, die den zu versorgenden Satz   */
/*                           variabler Laenge enthaelt.              */
/*            textp:   (IN)  Zeiger auf einen String, der den ein-   */
/*                           zutragenden Text enthaelt. Die Laenge   */
/*                           des Strings legt implizit die Laenge    */
/*                           des Satzes fest (Laenge String + 4).    */
/*                                                                   */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
static void
fill_buff(iedbuff *p,char *textp)
{
    size_t l_text;                          /* Laenge des String */
    if ((l_text = strlen(textp)) > 2044)
        l_text = 2044;       /* Laenge auf 2044 Zeichen begrenzen */

    strncpy((char *)p->text,textp,l_text);   /* Text eintragen */
    p->length = l_text + 4;                  /* Satzlaenge versorgen */
}

/*********************************************************************/
/* Funktion: edtcmd                                                  */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion traegt die uebergegebenen Strings in Saetze        */
/* variabler Laenge ein (DVS-Format) und ruft anschliessend die      */
/* CMD-Schnittstelle des EDT auf.                                    */
/*                                                                   */
/* Parameter: cmd:     (IN)  Zeiger auf einen String, der die        */
/*                           auszufuehrenden EDT-Anweisung(en)       */
/*                           enthaelt. Die Laenge des Strings legt   */
/*                           implizit die Laenge des Satze fest      */
/*                           (Laenge String + 4).                    */
/*            msg1:    (IN)  Zeiger auf einen String, der den ein-   */
/*                           zutragenden Text enthaelt. Die Laenge   */
/*                           des Strings legt implizit die Laenge    */
/*                           des Satzes fest (Laenge String + 4).    */
/*            msg2:    (IN)  Zeiger auf einen String, der den ein-   */
/*                           zutragenden Text enthaelt. Die Laenge   */
/*                           des Strings legt implizit die Laenge    */
```

```
/*                          des Satzes fest (Laenge String + 4).   */
/*                                                                 */
/* Rueckgabewert: keiner                                           */
/*******************************************************************/
static void
edtcmd(char *cmd,char *msg1,char *msg2)
{
    fill_buff(command,cmd);
    fill_buff(message1,msg1);
    fill_buff(message2,msg2);
    IEDTCMD(&glcb,&upcb,command,message1,message2);
}

/*******************************************************************/
/* Funktion: edtget                                                */
/*                                                                 */
/* Aufgabe:                                                        */
/* Diese Funktion liest mit der Funktion GET der Unterprogramm-    */
/* Schnittstelle eine Zeile aus der aktuellen Arbeitsdatei. Es wird */
/* relativ zum Satz mit Zeilennummer 0 gelesen.                    */
/*                                                                 */
/* Parameter: glcb:    (IN)  Zeiger auf den von EDT mitgelieferten */
/*                           glcb.                                 */
/*            rec:     (IN)  Zeiger auf einen Datenbereich, in dem */
/*                           die von der Funktion GET gelesene     */
/*                           Zeile abgelegt wird.                  */
/*            disp:    (IN)  Gibt an, die wievielte Zeile ab der   */
/*                           Zeilennummer 0 gelesen werden soll.   */
/*            key      (OUT) Zeiger auf ein Feld, in dem der gele- */
/*                           sene Schlüssel hinterlegt wird        */
/*******************************************************************/
static void
edtget(iedglcb *glcb,char *rec,int disp,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;
    iedbyte key1[8] = {0,0,0,0,0,0,0,0};

    /* Kontrollblock IEDAMCB versorgen */
    IEDAMCB_SET_NO_MARKS(amcb);
    amcb.length_key_outbuffer = 8;
    amcb.length_rec_outbuffer = 2044;
    amcb.length_key1 = 8;
    amcb.displacement = disp;  /* lese <disp> Sätze nach Nr. 0 */
    /* Nimm aktuelle Arbeitsdatei (aus glcb) */
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
    IEDTGET(glcb,&amcb,key1,key,rec);
    rec[amcb.length_rec] = 0;  /* set C-string end */
}
```

```
/*********************************************************************/
/* Funktion: edtput                                                  */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion schreibt mit der IEDTPUT-Funktion einen Satz an    */
/* die durch den Schlüssel key bezeichnete Stelle.                   */
/*                                                                   */
/* Parameter: glcb:    (IN)  Zeiger auf den von EDT mitgelieferten   */
/*                           glcb.                                   */
/*            rec:     (IN)  Zeiger auf einen Datenbereich, in dem   */
/*                           der zu schreibende Satz übergeben wird.*/
/*            key      (IN)  Zeiger auf ein Feld, in dem der zu      */
/*                           (über-)schreibende Schlüssel steht.     */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
static void
edtput(iedglcb *glcb,char *rec,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;

    amcb.length_key = 8;
    amcb.marks.mark_field = 0;
    amcb.length_rec = strlen(rec);
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
    IEDTPUT(glcb,&amcb,key,(unsigned char *)rec);
}

/* Statische Daten für die Verschlüsselung */
static char* lcc = "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz";
static char* ucc = "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ";

/*********************************************************************/
/* Funktion: ROT13                                                   */
/*                                                                   */
/* Aufgabe:                                                          */
/* Diese Funktion liest mit der Funktion EDTGET die gesamte          */
/* Arbeitsdatei, und verschlüsselt jede Zeile nach dem ROT13         */
/* Verfahren (d.h. alle Buchstaben werden um 13 Zeichen verschoben   */
/* Damit ergibt 2fache Verschlüsselung wieder das Original           */
/*                                                                   */
/* Parameter: glcb:    (IN)  Zeiger auf einen glcb, der für Aufrufe */
/*                           von IEDTGET oder IEDTPUT verwendet      */
/*                           wird.                                   */
/*            cmd:     (IN)  Argumente, mit denen ROT13 aufgerufen   */
/*                           wurde (nicht benutzt).                  */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
```

```
void
ROT13(iedglcb *glcb,iedbuff *cmd)
{
    char line[2048];
    iedbyte key[8];
    int disp;

    /* Lese alle Zeilen der aktuellen Arbeitsdatei */
    for (disp = 1; disp < 99999999; disp++)  /* -------------------- (1) */
    {
        unsigned int j = 0;

        /* Naechste Zeile lesen */
        edtget(glcb,(char *)&line,disp,key);
        if (glcb->IEDGLCB_RC_MAINCODE != 0)  /* -------------------- (2) */
        {
            glcb->IEDGLCB_RC_MAINCODE = IEDGLCBcmd_runtime_error;
           return; /* evtl. Meldung vom IEDTGET bleibt stehen */
        }

        /* Wenn jenseits der letzten Zeilennummer gelesen wird */
        /* gibt IEDTGET "last record" zurück */
        if (glcb->IEDGLCB_RC_SUBCODE1 == IEDGLCBlast_record)
            break; /* Schleife verlassen */
        /* Verschlüssele die Zeile */
        for (j = 0; j < strlen((char *)&line); j++)
        {
            char* pos;
            char ch = line[j];
            if (isalpha(ch))
            {
                if (islower(ch))
                    pos = index(lcc,ch);
                else
                    pos = index(ucc,ch);
                pos += 13;
                line[j] = *pos;
            }
        }
        /* Schreibe Zeile zurück */
        edtput(glcb,(char *)&line,key);
        if (glcb->IEDGLCB_RC_MAINCODE != 0)
        {
            glcb->IEDGLCB_RC_MAINCODE = IEDGLCBcmd_runtime_error;
            return; /* evtl. Meldung vom IEDTPUT bleibt stehen */
        }
    }
```

```
    /* setze Subcode zurück, damit keine Meldung kommt */
    glcb->IEDGLCB_RC_SUBCODE1 = 0;     /* ------------------------- (3) */
}

/*********************************************************************/
/* Funktion: ROT13@I                                                 */
/*                                                                   */
/* Aufgabe:                                                          */
/* Dies ist die Initialisierungsroutine zu ROT13.                    */
/*                                                                   */
/* Parameter: glcb:    (IN)  Zeiger auf einen glcb in dem u.a. der   */
/*                           beim Aufruf von ROT13 zu verwendende    */
/*                           Zeichensatz festgelegt werden kann.     */
/*                                                                   */
/* Rueckgabewert: keiner                                             */
/*********************************************************************/
void
ROT13@I(iedglcb *glcb)
{
    glcb->indicator.compatible_format = 1;
    memcpy(glcb->ccsn,"EDF04F  ",8);   /* ROT13 erwartet EDF04F */
    glcb->rc.rc_nbr = 0;
}

/*********************************************************************/
/* Hauptprogramm                                                     */
/*********************************************************************/
int
main(int argc,char *argv[])
{
    char cmd[257];      /* Bereich zum Aufbau von EDT-Anweisungen */
    int opt;
    extern int optind, opterr, optopt;
    extern char *optarg;

    /* Aufrufparameter auswerten */
    while ((opt = getopt(argc,argv,"f:x:")) != -1)    *------------- (4) */
    {
        switch (opt)
        {
        case 'f':
            sprintf(cmd,"@OPEN FILE=%s",optarg);
            break;
        case 'x':
            sprintf(cmd,"@OPEN POSIX-FILE=%s",optarg);
            break;
        case ':':
            printf("Argument für -%c fehlt",optopt);
```

```
            return 0;
        default:
            printf("Aufruf mit -f <file> oder -x <posix-file>");
            return 0;
        }
    }

    /* buffer bereitstellen */
    command = (iedbuff *)malloc(2048);
    message1 = (iedbuff *)malloc(2048);
    message2 = (iedbuff *)malloc(2048);

    edtcmd(&(cmd[0]),"","");
    printrc("Fehler beim Einlesen der Datei.");

    /* Benutzeranweisung einrichten. MODLIB muss nicht gegeben werden.*/
    edtcmd("USE COM='*',ENTRY=ROT13","","");    /* ----------------- (5) */
    printrc("Fehler bei der @USE-Anweisung.");

    edtcmd("SETF(0);DIALOG","Beispiel 4 fuer die UP-Schnittstelle","");
    printrc("Fehler bei der @SETF- oder der @DIALOG-Anweisung!");

    edtcmd("HALT","","");
    return 0;                                   /* -------------- (6) */
}
```

Erläuterungen:

(1) Die hier implementierte Methode zum sequenziellen Lesen der Arbeitsdatei ist nicht sonderlich effektiv, da immer wieder von Beginn an gelesen wird. Man überlege sich, wie man unter Verwendung des in `key` zurück gelieferten Schlüssels mit konstantem `amcb.displacement = +1` einen günstigeren Algorithmus implementieren kann.

(2) Die Fehlerbehandlung wurde aus Gründen der Übersichtlichkeit nur rudimentär implementiert. Man sollte hier zumindest noch überprüfen, ob der EDT eine Meldung zurückgegeben hat und ggf. selbst eine Meldung (mit dem Original-Returncode des EDT eintragen).

(3) Ein unerwarteter Returncode führt immer zur Ausgabe der Meldung `EDT5410` nach Rückkehr aus der Anwenderroutine.

(4) Es werden die Aufrufparameter `-f <dateiname>` und `-x <posix-dateiname>` akzeptiert.

(5) Die Benutzeranweisung wird schon vor dem Wechsel in den Bildschirmdialog eingerichtet. Da der Entry im Programm selbst liegt, kann man auf Angabe einer Bibliothek verzichten. Da die Funktion `ROT13` den übergebenen Anweisungsstring nicht auswertet, kann man sie beliebig, z.B. mit `*rot` oder auch nur mit `*` aufrufen. Ein Aufruf mit @RUN E=ROT13 ist daher ebenfalls möglich.

(6) Wenn das Programm "produktiv" verwendet werden soll, muss sich hier die Überprüfung auf evtl. noch geöffnete Dateien und ein entsprechender Beendigungsdialog anschließen.

Wenn die in Abschnitt „Produktion von Hauptprogrammen in C“ auf Seite 103 erklärte Prozedur im BS2000 in einer Datei namens `CC.DO` und die Quelldatei als S-Element `BEISPIEL4.C` in der Bibliothek `EDT.BEISPIELE` abgelegt ist, kann das obige Programm mit

```
/CALL-PROC CC.DO,(4)
```

übersetzt und gebunden werden. Das erzeugte Prograng ist anschließend z.B. mit

```
/ST-EX-P (edt.beispiele,bsp4c),p-p='-f edt.test4'
```

ausführbar. Hier wurde bewusst extrem abgekürzt, um zu zeigen, dass mit einem solchen Starterprogramm auch im BS2000 relativ bequem der EDT direkt zum Editieren einer bestimmten Datei gestartet werden kann.

Bei Ablauf von `CC.DO` werden etwa folgende Ausgaben vom System bzw. vom Compiler erzeugt:

```
%  BLS0524 LLM 'SDFCC', VERSION '03.1A40' OF '2005-02-03 16:16:36' LOADED
%  BLS0551 COPYRIGHT (C) Fujitsu Siemens Computers GmbH 2005. ALL RIGHTS
RESERVED
%  CDR9992 : BEGIN C/C++(BS2000/OSD) VERSION 03.1A40
%  CDR9907 : NOTES: 1  WARNINGS: 0  ERRORS: 0  FATALS: 0 -------------- (1)
%  CDR9937 : MODULES GENERATED, CPU TIME USED = 3.3800 SEC
%  BND3102 SOME WEAK EXTERNS UNRESOLVED
%  BND1501 LLM FORMAT: '1'
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED EXTERNAL'
%  CDR9936 : END; SUMMARY: NOTES: 1  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CCM0998 CPU TIME USED: 4.6826 SECONDS
```

Erläuterungen

(1) Die `"NOTE"` kommt zustande, weil der Aufrufparameter `cmd` von `ROT13` nicht angesprochen wird.

Die von diesem Beispielprogramm zu bearbeitende Datei `EDT.TEST4` habe den CCS `EDF04F` und sehe folgendermaßen aus:

```
Dies ist eine EDF04F Datei.
Sie enthält alle möglichen Sonderzeichen: ÀÁ¿çÆ.
Natürlich auch das € und das Å.
```

Dann erscheint nach Aufruf des Programms der folgende Bildschirm:

```
   1.00 Dies ist eine EDF04F Datei<·····································
   2.00 Sie enthält alle möglichen Sonderzeichen:ÀÁ¿çÆ<·····················
   3.00 Natürlich auch das € und das Å<·····································
   4.00 ····································································
   5.00 ····································································
   6.00 ····································································
   7.00 ····································································




*rot·····································································0001.00:00001(00)
```

Nach Eingabe der Benutzeranweisung `*rot` sieht die Ausgabe wie folgt aus:

```
   1.00 Qvrf vfg rvar RQS04S Qngrv<·····································
   2.00 Fvr raguäyg nyyr zötyvpura Fbaqremrvpura: ÀÁ¿çÆ<·····················
   3.00 Angüeyvpu nhpu qnf € haq qnf Å<·····································
   4.00 ····································································
   5.00 ····································································
   6.00 ····································································
   7.00 ····································································




·····································································0001.00:00001(00)
```

7.5 Beispiel 5 - Assembler-Hauptprogramm

Beispiel 5 ist funktionell identisch zu Beispiel 1, nur dass statt in C in Assembler programmiert wurde.

```
          TITLE 'BEISPIEL5'
*********************************************************************
*                                                                   *
* Beispiel 5                                                        *
*                                                                   *
* Dieses Beispiel verwendet ausschliesslich die iedtcmd-            *
* Schnittstelle zur Ausfuehrung von EDT-Anweisungen.                *
* Die Funktionalität ist identisch mit der des "Beispiel 1" in C.   *
*                                                                   *
* Das Beispielprogramm fuehrt folgende Aktionen durch:              *
*                                                                   *
* 1) Einlesen eines Auswahlkriteriums (CCSN)                        *
* 2) Ausgabe eines Inhaltsverzeichnisses in die Arbeitsdatei 0      *
*    mit der @SHOW-Anweisung (Format 1).                            *
* 3) Loeschen aller Zeilen, die nicht das Auswahlkriterium          *
*    enthalten mit der @ON-Anweisung (Format 10).                   *
* 4) Einstellen der Arbeitsdatei 0 und anschliessender Wechsel      *
*    in den F-Modus-Dialog mit einer @SETF- und einer nach-         *
*    folgenden @DIALOG-Anweisung.                                   *
* 5) Der Anwender kann nun die ausgegebenen Zeilen editieren        *
*    und schliesslich den EDT und damit auch diese Beispiel-        *
*    programm beenden.                                              *
*                                                                   *
*********************************************************************
*
BEISP5   START
BEISP5   AMODE ANY
BEISP5   RMODE ANY
         GPARMOD 31
*
         BALR  R10,0
         USING *,R10
*
*        STARTMELDUNG AUSGEBEN
*
         LA    R1,6               * LAENGE = 6 FUER DIE AUSGABE
         STH   R1,LEERMSG         * EINES LEERZEICHENS
*
         WROUT LEERMSG,WROUTERR
         WROUT STARTMSG,WROUTERR
         WROUT LEERMSG,WROUTERR
```

```
*
*        CCSN EINLESEN
*
         WRTRD PROMPT,,EINB,,12,WRTRDERR
*
         LH    R15,EINB           LAENGE DER EINGABE
         SH    R15,=Y(5)          4 BYTE FUER SLF UND 1 FUER EX
         EX    R15,EXMVCCSN       CCS IN DIE ON-ANWEISUNG UEBERTRAGEN
         LA    R14,CCSN           AUF ZEICHEN NACH DEM CCSN
         LA    R14,1(R14,R15)     POSITIONIEREN
         MVI   0(R14),''''        APOSTROPH SETZEN     ---------------- (1)
*
         LA    R1,4               * LAENGE = 4 FUER DIE UEBERGABE
         STH   R1,LEERMSG         * EINES LEERSTRINGS AN DEN EDT
*
*        INHALTSVERZEICHNIS IN DIE ARBEITSDATEI 0 AUSGEBEN
*
         LA    R1,CMD1            ADRESSE EDT-ANWEISUNG
         LA    R2,LEERMSG         MSG1 = LEERSTRING
         LA    R3,LEERMSG         MSG2 = LEERSTRING
         LA    R4,ZUSMSG1         ADR. DER AUSZUGEBENDEN FEHLERMELDUNG
*
         BAL   R11,CMDCALL        AUFRUF DER CMD-SCHNITTSTELLE DES EDT
*
*       ALLE ZEILEN LOESCHEN, DIE NICHT DEM SUCHKRITERIUM (CCSN)
*       ENTSPRECHEN UND DIE VERBLIEBENEN ZEILEN NEU DURCHNUMMERIEREN
*
         LA    R1,CMD2            ADRESSE EDT-ANWEISUNG
         LA    R2,LEERMSG         MSG1 = LEERSTRING
         LA    R3,LEERMSG         MSG2 = LEERSTRING
         LA    R4,ZUSMSG2         ADR. DER AUSZUGEBENDEN FEHLERMELDUNG
*
         BAL   R11,CMDCALL        AUFRUF DER CMD-SCHNITTSTELLE DES EDT
*
*        IN DIE ARBEITSDATEI 0 WECHSELN UND IN DEN
*        F-MODUS-DIALOG UMSCHALTEN
*
         LA    R1,CMD3            ADRESSE EDT-ANWEISUNG
         LA    R2,MESSAGE3        ADRESSE DER AUSZUGEBENDEN MELDUNG
         LA    R3,LEERMSG         MSG2 = LEERSTRING
         LA    R4,ZUSMSG3         ADR. DER AUSZUGEBENDEN FEHLERMELDUNG
*
         BAL   R11,CMDCALL        AUFRUF DER CMD-SCHNITTSTELLE DES EDT
*
```

```
*        ENDEMELDUNG AUSGEBEN
*
         LA    R1,6               * LAENGE = 6 FUER DIE AUSGABE
         STH   R1,LEERMSG         * EINES LEERZEICHENS
*
         WROUT LEERMSG,WROUTERR
         WROUT ENDEMSG,WROUTERR
         WROUT LEERMSG,WROUTERR
*
         TERM                           PROGRAMM BEENDEN
*
*        BEHANDLUNG VON EDT-FEHLERN
*
EDTERR   EQU   *
         OC    EGLRMSGL,EGLRMSGL        LAENGE DER EDT-MELDUNG = 0 ?
         BZ    NOMSG                    JA, DANN NICHTS AUSGEBEN
*
*        ZUSAETZLICH ANGEGEBENE MELDUNG AUSGEBEN
*
         LH    R5,0(R4)                 LAENGE DER ZUSAETZLICHEN MELDUNG
         BCTR  R5,0                     -1 FUER EX-BEFEHL
         EX    R5,EXMVC                 MELDUNG UEBERNEHMEN
*
         WROUT ZUSMSG,WROUTERR          ZUSAETZLICHE MELDUNG AUSGEBEN
*
*        EDT-FEHLERMELDUNG AUSGEBEN
*
         LH    R1,EGLRMSGL              * LAENGE DER EDT-MELDUNG
         LA    R1,5+14(R1)              * + 5 (WEGEN SATZLAENGENFELD)
*                                       * + 14 (WG. "MELDUNGSTEXT: ")
         STH   R1,ERRMSG1               * IM LAENGENFELD ABLEGEN
         MVC   ERRTEXT,EGLRMSGF         MELDUNGSTEXT UEBERTRAGEN
*
         WROUT ERRMSG1,WROUTERR         EDT-MELDUNG AUSGEBEN
*
NOMSG    EQU   *
         TERM                           PROGRAMM BEENDEN
*
*        BEHANDLUNG VON WRTRD-FEHLERN
*
WRTRDERR EQU   *
         WROUT ERRMSG2,WROUTERR
*
WROUTERR EQU   *
         TERM                           PROGRAMM BEENDEN
         EJECT
```

```
*********************************************************************
*                                                                   *
*        UNTERPROGRAMME                                             *
*                                                                   *
*********************************************************************
*
*********************************************************************
* UNTERPROGRAMM: CMDCALL                                            *
*                                                                   *
* AUFGABE:                                                          *
* DIESES UNTERPROGRAMM VERSORGT DIE PARAMETERLISTE FUER DIE CMD-    *
* SCHNITTSTELLE DES EDT UND RUFT ANSCHLIESSEND DIE CMD-SCHNITTSTELLE *
* AUF. NACH DER RUECKKEHR VOM EDT WIRD IM OK-FALL ZUM AUFRUFER      *
* ZURUECKGEKEHRT, IM FEHLERFALL ZU EINER FEHLERROUTINE, WELCHE      *
* DIE ZUSAETZLICH UEBERGEBENE FEHLERMELDUNG SOWIE DIE VOM EDT       *
* ZURUECK GELIEFERTE MELDUNG AUSGIBT.                               *
*                                                                   *
* PARAMETER: (R1):  (IN) ADRESSE EINES SATZES VARIABLER LAENGE      *
*                        (DVS-FORMAT), DER DIE AUSZUFUEHRENDEN      *
*                        EDT-ANWEISUNGEN ENTHAELT.                  *
*            (R2):  (IN) ADRESSE EINES SATZES VARIABLER LAENGE      *
*                        (DVS-FORMAT), DER DIE VOM EDT AUSZUGEBENDE *
*                        MELDUNG1 ENTHAELT.                         *
*            (R3):  (IN) ADRESSE EINES SATZES VARIABLER LAENGE      *
*                        (DVS-FORMAT), DER DIE VOM EDT AUSZUGEBENDE *
*                        MELDUNG2 ENTHAELT.                         *
*            (R4):  (IN) ADRESSE EINES SATZES VARIABLER LAENGE      *
*                        (DVS-FORMAT), DER DIE IM FEHLERFALL        *
*                        ZUSAETZLICH AUSZUGEBENDE MELDUNG ENTHAELT. *
*            (R11): (IN) RUECKSPRUNGADRESSE                         *
*                                                                   *
* RUECKGABEWERT: KEINER                                             *
*********************************************************************
*
CMDCALL  EQU   *
         STM   R1,R3,COMMAND           CMD-PARAMETERLISTE VERSORGEN
         LA    R1,CMDPL                ADR. DER CMD-PARAMETERLISTE
         LA    R13,SAVEAREA            ADR. DER SAVEAREA
         L     R15,=V(IEDTCMD)         ADR. DER CMD-SCHNITTSTELLE
*
         BALR  R14,R15                 EDT-CMD-SCHNITTSTELLE AUFRUFEN
*
         CLC   EGLMRET,=AL2(EUPRETOK)  FEHLER BEIM AUFRUF DES EDT ?
         BNE   EDTERR                  JA, FEHLERMELDUNGEN AUSGEBEN
         BR    R11                     NEIN, ZURUECK ZUM AUFRUFER
         EJECT
```

```
*************************************************************************
*                                                                       *
*        KONSTANTEN                                                     *
*                                                                       *
*************************************************************************
*
*        REGISTERDEFINITIONEN
*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
*************************************************************************
*                                                                       *
*        FELDER                                                         *
*                                                                       *
*************************************************************************
*
SAVEAREA DS    18F
*
EXMVC    MVC   ZUSMSG(0),0(R4)         EX-BEFEHL ZUR UEBERTRAGUNG
*                                      DER ZUSATZMELDUNG
*
EXMVCCSN MVC   CCSN(0),EINGABE         CCSN IN ON-ANWEISUNG EINTRAGEN
LEERMSG  DC    Y(LEERMEND-LEERMSG)     LEERER SATZ
         DS    CL2
         DC    ' '
         DC    ' '
LEERMEND EQU   *
*
PROMPT   DC    Y(PRMPTEND-PROMPT)      EINGABE-AUFFORDERUNG
         DS    CL2
         DC    C' '
         DC    C'Bitte CCSN eingeben (UTFE, UTF16, EDF...): '
PRMPTEND EQU   *
```

```
*
EINB     DC    Y(EINBEND-EINB)         EINGABEBEREICH FUER WRTRD
         DS    CL2
EINGABE  DC    C'        '
EINBEND  EQU   *
*
CMD1     DC    Y(CMD1END-CMD1)         EDT-ANWEISUNGSFOLGE 1
         DS    CL2
         DC    C'SHOW F=* TO 1 LONG'
CMD1END  EQU   *
*
CMD2     DC    Y(CMD2END-CMD2)         EDT-ANWEISUNGSFOLGE 2
         DS    CL2
         DC    C'ON &&:100-107: FIND NOT '''
CCSN     DC    C'        '
         DC    C'  DELETE;RENUMBER'
CMD2END  EQU   *
*
CMD3     DC    Y(CMD3END-CMD3)         EDT-ANWEISUNGSFOLGE 3
         DS    CL2
         DC    C'SETF(0);DIALOG'
CMD3END  EQU   *
*
MESSAGE3 DC    Y(MSG3END-MESSAGE3)     VOM EDT AUSZUGEBENDE MELDUNG
         DS    CL2
         DC    C'Beispiel 5 fuer die UP-Schnittstelle'
MSG3END  EQU   *
*
ERRMSG1  DC    Y(ERRM1END-ERRMSG1)     SATZ ZUR AUSGABE DER EDT-MELDUNG
         DS    CL2
         DC    C' '
         DC    C'Meldungstext: '
ERRTEXT  DS    CL80
ERRM1END EQU   *
*
ERRMSG2  DC    Y(ERRM2END-ERRMSG2)     FEHLERMELDUNG BEI EINGABEFEHLERN
         DS    CL2
         DC    C' '
         DC    C'Eingabe zu lang!'
ERRM2END EQU   *
*
STARTMSG DC    Y(STRTMEND-STARTMSG)    STARTMELDUNG
         DS    CL2
         DC    C' '
         DC    C'Start Beispiel5'
STRTMEND EQU   *
*
ENDEMSG  DC    Y(ENDEMEND-ENDEMSG)     ENDEMELDUNG
```

```
         DS    CL2
         DC    C' '
         DC    C'Ende Beispiel5'
ENDEMEND EQU   *
*
ZUSMSG1  DC    Y(ZUSM1END-ZUSMSG1)     ZUSAETZLICH AUSZUGEBENDE
         DS    CL2                     FEHLERMELDUNG BEI DER
         DC    C' '                    EDT-ANWEISUNGSFOLGE 1
         DC    C'Fehler bei der @SHOW-Anweisung!'
ZUSM1END EQU   *
*
ZUSMSG2  DC    Y(ZUSM2END-ZUSMSG2)     ZUSAETZLICH AUSZUGEBENDE
         DS    CL2                     FEHLERMELDUNG BEI DER
         DC    C' '                    EDT-ANWEISUNGSFOLGE 2
         DC    C'Fehler bei der @ON- oder der @RENUMBER-Anweisung!'
ZUSM2END EQU   *
*
ZUSMSG3  DC    Y(ZUSM3END-ZUSMSG3)     ZUSAETZLICH AUSZUGEBENDE
         DS    CL2                     FEHLERMELDUNG BEI DER
         DC    C' '                    EDT-ANWEISUNGSFOLGE 3
         DC    C'Fehler bei der @SETF- oder der @DIALOG-Anweisung!'
ZUSM3END EQU   *
*
ZUSMSG   DC    Y(ZUSMEND-ZUSMSG)       SATZ VARIABLER LAENGE ZUR
         DS    CL2                     AUSGABE DER ZUSAETZLICHEN
         DC    C' '                    FEHLERMELDUNG
         DC    CL80' '
ZUSMEND  EQU   *
*
*        PARAMETERLISTE DER EDT-CMD-SCHNITTSTELLE
*
CMDPL    DC    A(EDTGLCB)              ADRESSE EDTGLCB
         DC    A(EDTUPCB)              ADRESSE EDTUPCB
COMMAND  DC    A(0)                    ADRESSE DER EDT-ANWEISUNGEN
MSG1     DC    A(0)                    ADRESSE MESSAGE1
MSG2     DC    A(0)                    ADRESSE MESSAGE2
*
*        EDT-SPEZIFISCHE SCHNITTSTELLEN-MAKROS DER V17.0A
*
         IEDTGLCB C,VERSION=2          ------------------------------- (2)
*
         IEDTUPCB C,VERSION=3
         END
```

Erläuterungen

(1) Zwischen CCSN und Apostroph dürfen keine Leerzeichen stehen, sonst ist eine Suche nach Teilzeichenketten (z.B. EDF) nicht möglich.

(2) Es wird die V17-Version der jeweiligen Schnittstelle generiert.

Wenn die in Abschnitt „Produktion von Hauptprogrammen in Assembler“ auf Seite 108 erklärte Prozedur im BS2000 in einer Datei namens ASS.DO und die Quelldatei als S-Element BEISPIEL5.ASS in der Bibliothek EDT.BEISPIELE abgelegt ist, kann das obige Programm mit

```
/CALL-PROC ASS.DO,(5)
```

übersetzt und gebunden werden. Das erzeugte Programm ist anschließend mit

```
/START-EXECUTABLE-PROGRAM (E=BSP5A,L=EDT.BEISPIELE)
```

ausführbar. Bei Ablauf von ASS.DO werden etwa folgende Ausgaben vom System bzw. vom Assembler erzeugt:

```
%  BLS0523 ELEMENT 'ASSEMBH', VERSION '012', TYPE 'C' FROM LIBRARY' :MARS:
$TSOS.SYSPRG.ASSEMBH.012' IN PROCESS
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '01.2C00' OF '2002-03-06' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 2002. ALL RIGHTS
RESERVED
%  ASS6010 V01.2C00 OF BS2000 ASSEMBH  READY
%  ASS6011 ASSEMBLY TIME: 836 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 291 MSEC
%  ASS6012 END OF ASSEMBH
%  BND0500 BINDER VERSION 'V02.3A00' STARTED
%  BND1501 LLM FORMAT: '1'
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'
```

Die von diesem Beispielprogramm erzeugten Ausgaben entsprechen denen von Beispiel1, mit Ausnahme der Meldung CCM0998 nach Programmbeendigung.

7.6 Beispiel 6 - Assembler-Anwendungsroutine

Dieses Beispiel benutzt die Schnittstellen IEDTGTM, IEDTPARL und IEDTEXE, um aus der aktuellen Arbeitsdatei markierte Dateinamen zu lesen und die dazugehörigen Dateien nacheinander in freie Arbeitsbereiche einzulesen.

```
          TITLE 'BEISPIEL6'
*********************************************************************
*                                                                   *
* Beispiel 6                                                        *
*                                                                   *
* Dieses Beispiel realisiert eine Anwenderroutine, die das Ein-     *
* lesen mehrerer Dateien erlaubt, die in einer Dateinamensliste     *
* markiert wurden.                                                  *
* Es verwendet die iedtgtm-Schnittstelle, um alle markierten Sätze  *
* aus dem aktuellen Arbeitsbereich zu lesen.                        *
* Es wird erwartet, dass die Sätze Dateinamen enthalten, die z.B.   *
* mit SHOW F=* TO 1 erstellt wurden.                                *
*                                                                   *
* Das Beispielprogramm fuehrt folgende Aktionen durch:              *
*                                                                   *
* 1) In Schleife werden alle markierten Sätze der aktuellen         *
*    Arbeitsdatei gelesen.                                          *
* 2) Für jede markierte Datei wird über einen Aufruf von iedtparl   *
*    eine freie Arbeitsdatei gesucht.                               *
* 3) Falls es noch eine freie Arbeitsdatei gibt, wird die Datei     *
*    mittels @COPY FILE= (via iedtexe) eingelesen                   *
*                                                                   *
*********************************************************************
*
CMULTI   CSECT
CMULTI   AMODE ANY
CMULTI   RMODE ANY
         GPARMOD 31
*
         STM   R14,R12,12(R13)    * REGISTER SICHERN
         LR    R10,R15            * BASISREGISTER VERSORGEN
         USING CMULTI,R10
*
*        VERSORGE PL MIT DEM MITGELIEFERTEN GLCB
*
         L     R11,0(,R1)         GLCB VON EDT
         ST    R11,EXEPL          -> IN DIE EXE PL
         ST    R11,PARLPL         -> IN DIE PARL PL
         ST    R11,GTMPL          -> IN DIE GTM PL
         USING EDTGLCB,R11
*
```

```
*        SAVEAREA FUER UP-AUFRUFE VERSORGEN
*
         LR    R7,R13             SICHERE R13
         LA    R13,SAVEAREA
*
*        AMCB  FUER GTM VERSORGEN
*
         MVC   EAMFILE,EGLFILE    AKTUELLE ARBEITSDATEI AUS GLCB
         MVC   EAMDISP,=A(1)      LESE SATZ NACH SCHLUESSEL
         MVC   EAMLKEY1,=Y(8)     LAENGENFELDER VERSORGEN
         MVC   EAMPKEY,=Y(8)
         MVC   EAMPREC,=Y(54)
*
*        AMCB  FUER PARL VERSORGEN
*
         MVC   PAMFILE(3),=C'LOO' ARBEITSDATEI FUER PARL
         MVC   PAMLKEY1,=Y(8)     LAENGENFELDER VERSORGEN
         MVC   PAMPKEY,=Y(8)
         MVC   PAMPREC,=Y(EPLPARLL)
*
*        SCHLEIFE UEBER ALLE MARKIERTEN SAETZE
*
         LA    R3,0(0,0)          ZAEHLER FUER ARBEITSDATEIEN
LOOP     DS    0Y
         LA    R1,GTMPL           ADRESSE EDT-ANWEISUNG
         L     R15,=V(IEDTGTM)    GTM ROUTINE
         BALR  R14,R15
         CLC   EGLMRET,=Y(EAMRETOK) GTM OK ?
         LA    R1,ERRGTM          FEHLERMELDUNG
         BNE   LOOPERR            FEHLERAUSGANG
         CLI   EGLSR1,EAMOK12     LETZTER MARKIERTER SATZ?
         BE    LOOPEX             NORMALER AUSGANG
*
*        SUCHE FREIE ARBEITSDATEI
*
LOOPI    DS    0Y
         LA    R4,ARBDATNR        ABDRUCKBARE NUMMERN
         LA    R4,0(R3,R4)        ADDIERE ZAEHLER
         MVC   PAMFILE+1(2),0(R4) NR IN DEN AMCB UEBERTRAGEN
         LA    R1,PARLPL          FUER PARL AUFRUF
         L     R15,=V(IEDTGET)    GET ROUTINE (PARL)
         BALR  R14,R15
         CLC   EGLMRET,=Y(EAMRETOK)   PARL OK?
         LA    R1,ERRPARL
         BNE   LOOPERR            FEHLERAUSGANG
         CLI   EPLEMPTY,'1'       ARBEITSDATEI LEER?
         BE    LOOPIEX            SCHLEIFE VERLASSEN
```

```
         LA    R3,2(,R3)          NAECHSTE ARBEITSDATEI IN 2ER SCHR.
*                                 DA 2STELLIG IN ARBDATNR
         CH    R3,=Y(44)          LETZTE ARBEITSDATEI ERREICHT?
         BH    LOOPEX             NORMALER AUSGANG
         B     LOOPI              NAECHSTE VERSUCHEN
*
LOOPIEX  DS    OY
         MVC   ADAT,PAMFILE+1     ARBEITSDATEI IN SETF-ANWEISUNG
         MVI   FILE,' '           DATEINAMEN VORLOESCHEN
         MVC   FILE+1(53),FILE
         LH    R15,EAMLREC        LAENGE DES GELESENEN SATZES
         BCTR  R15,0              MINUS 1 FUER EX
         EX    R15,EXMVCFIL       DATEINAME IN @COPY-ANWEISUNG
         LA    R1,EXEPL           FUER EXE AUFRUF
         L     R15,=V(IEDTEXE)
         BALR  R14,R15
         CLC   EGLMRET,=Y(EUPRETOK)   EXE OK?
         LA    R1,ERREXE
         BNE   LOOPERR            FEHLERAUSGANG
         MVC   KEY1(8),KEY        NEUE BASIS IST DER GELESENE SATZ --- (1)
         B     LOOP
*
LOOPEX   DS    OY
         MVC   EGLMRET,=Y(EUPRETOK)   RC OK
         MVI   EGLSR1,EUPOKOO
         LR    R13,R7             R13 WIEDERHERSTELLEN
         LM    R14,R12,12(R13)
         BR    R14                ZURÜCK ZUM EDT
*
LOOPERR  DS    OY                              ---------------------- (2)
         MVC   EGLMRET,=Y(EUPRTERR)   FEHLER IM BENUTZERPROGRAMM
         MVI   EGLSR1,EUPOKOO
         MVC   EGLRMSG(ERRMSGL+2),0(R1)
         LR    R13,R7             R13 WIEDERHERSTELLEN
         LM    R14,R12,12(R13)
         BR    R14
*********************************************************************
*        INITIALISIERUNGS-ROUTINE                                   *
*********************************************************************
         ENTRY CMULTI@I
CMULTI@I DS    OD                 -------------------------------- (3)
         STM   R14,R12,12(R13)
         USING CMULTI@I,R15
         L     R11,0(,R1)         ADRESSE DES GLCB
         USING EDTGLCB,R11
         MVC   EGLCCSN,=C'EDF041  '
         LM    R14,R12,12(R13)
         BR    R14
```

```
         DROP  R11,R15
         EJECT
*************************************************************************
*                                                                       *
*        KONSTANTEN                                                     *
*                                                                       *
*************************************************************************
*
*        REGISTERDEFINITIONEN
*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
*************************************************************************
*                                                                       *
*        FELDER                                                         *
*                                                                       *
*************************************************************************
SAVEAREA DS    18F                    SAVEAEREA
*
EXMVCFIL MVC   FILE(0),REC            DATEINAME IN COPY-ANWEISUNG
*
REC      DS    CL54                   BEREICH FUER DATEINAMEN
*
ERRGTM   DC    Y(ERRMSGL)
         DC    'FEHLER BEI IEDTGTM   '
ERRMSGL  EQU   *-ERRGTM-2
ERRPARL  DC    Y(ERRMSGL)
         DC    'FEHLER BEI IEDTPARL  '
ERREXE   DC    Y(ERRMSGL)
         DC    'FEHLER BEI IEDTEXE   '
```

```
*
CMD1     DC    Y(CMD1END-CMD1)         EDT-ANWEISUNG: COPY
         DS    CL2
         DC    C'@SETF('
ADAT     DC    C'00'
         DC    ');@COPY FILE='
FILE     DC    CL54' '
CMD1END  EQU   *
*
ARBDATNR DC    C'000102030405060708091012131415161718192021 22'
*
*        PARAMETERLISTE DER EDT-EXE-SCHNITTSTELLE
*
EXEPL    DC    A(0)                    ADRESSE EDTGLCB
         DC    A(CMD1)                 ADRESSE DER ANWEISUNG
*
*        PARAMETERLISTE DER EDT-GTM-SCHNITTSTELLE
*
GTMPL    DC    A(0)                    ADRESSE EDTGLCB
         DC    A(EDTAMCB)              ADRESSE EDTAMCB
         DC    A(KEY1)                 ADRESSE KEY1 (IN)
         DC    A(KEY)                  ADRESSE KEY  (OUT)
         DC    A(REC)                  ADRESSE DES GELESENEN SATZES
*
*        PARAMETERLISTE DER EDT-PARL-SCHNITTSTELLE
*
PARLPL   DC    A(0)                    ADRESSE EDTGLCB
         DC    A(PEDTAMCB)             ADRESSE EDTAMCB
         DC    A(KEY1)                 ADRESSE KEY1 (IN)
         DC    A(KEY)                  ADRESSE KEY  (OUT)
         DC    A(EDTPARL)              ADRESSE DER AUSGABE INFO
*
KEY1     DC    2A(0)                   SCHLUESSEL FUER GTM
KEY      DC    2A(0)                   SCHLUESSEL (RUECKGABEWERT)
*
*        EDT-SPEZIFISCHE SCHNITTSTELLEN-MAKROS DER V17.0A
*
         IEDTAMCB C,VERSION=2
*
         IEDTAMCB C,P,VERSION=2
*
         IEDTPARL C,VERSION=4
*
         IEDTGLCB D,VERSION=2
CMULTI   CSECT
         END
```

Erläuterungen

(1) EDTGTM liest ausgehend vom angegebenen Schlüssel in der durch EAMDISP bestimmten Richtung (hier vorwärts). Daher muss der gelesene Schlüssel zum neuen Ausgangspunkt gemacht werden.

(2) Die Fehlerbehandlung ist aus Gründen der Übersichtlichkeit nur angedeutet. Man sollte zumindest den Original-Returncode, der von der Schnittstelle geliefert wurde, aufbereiten und ausgeben.

(3) Durch Angabe der Initialisierungsroutine wird gleichzeitig erreicht, dass CMULTI mit dem V17-GLCB gerufen wird.

Wenn die in Abschnitt „Produktion von Anwenderroutinen in Assembler“ auf Seite 110 erklärte Prozedur im BS2000 in einer Datei namens ASSMOD.DO und die Quelldatei als S-Element ANWEND2.ASS in der Bibliothek EDT.BEISPIELE abgelegt ist, kann das obige Programm mit

```
/CALL-PROC ASSMOD.DO,(2)
```

übersetzt und gebunden werden. Das erzeugte Programm ist anschließend aus dem EDT heraus mit

@USE COMMAND='*',ELEMENT=CMULTI,MODLIB=EDT.BEISPIELE

ladbar. Die Prozedur ASSMOD.DO erzeugt in etwa folgende Ausgaben:

```
%  BLS0523 ELEMENT 'ASSEMBH', VERSION '012', TYPE 'C' FROM LIBRARY ':MARS:
$TSOS.SYSPRG.ASSEMBH.012' IN PROCESS
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '01.2C00' OF '2002-03-06' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 2002. ALL RIGHTS
RESERVED
%  ASS6010 V01.2C00 OF BS2000 ASSEMBH  READY
%  ASS6011 ASSEMBLY TIME: 480 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 12 MSEC
%  ASS6012 END OF ASSEMBH
```

Die Arbeitsweise der Routine sei im L-Modus demonstriert, d.h. es ist vorausgesetzt, dass der EDT schon geladen ist und auf eine Eingabe im L-Modus wartet:

```
  1.     @SHOW F=* TO 1                                ---------- (1)
550.     @O&F'BEISPIEL'                                ---------- (2)
550.     @USE COM='*',E=CMULTI,M=EDT.BEISPIELE         ---------- (3)
550.     *CMULTI                                       ---------- (4)
  %  EDT5999 FEHLER BEI IEDTEXE                       ---------- (5)
  1. @PROC 22
  1. @STA=PAR TO 1                                     ---------- (6)
254. @ON & P '$ ='
  7.0000     % =    1.0000  $ =  549.0000  * =  550.0000  ? =   22.0000
 18.0000     % =    1.0000  $ =  309.0000  * =  310.0000  ? =    0.0000
 29.0000     % =    1.0000  $ =  187.0000  * =  188.0000  ? =    0.0000
 40.0000     % =    1.0000  $ =  179.0000  * =  180.0000  ? =    0.0000
 51.0000     % =    1.0000  $ =  235.0000  * =  236.0000  ? =    0.0000
 62.0000     % =    1.0000  $ =    1.0000  * =    1.0000  ? =    0.0000
....
```

Erläuterungen

(1) Die Dateiliste wird in der Arbeitsdatei aufgebaut. Die Liste umfasst hier z.B. `549` Einträge.

(2) Alle Dateien, die den Namensbestandteil `BEISPIEL` haben, werden gesucht und markiert.

(3) Die Anwenderroutine `CMULTI` wird aus der Bibliothek `EDT.BEISPIELE` geladen und soll Benutzeranweisungen bearbeiten, die mit `'*'` beginnen.

(4) Die Anwenderroutine wird aufgerufen. Da sie den Eingabestring nicht auswertet, könnte man sie auch mit einer beliebigen anderen Anweisung, z.B. `*XXX` aufrufen.

(5) Die Meldung stammt von der Anwenderroutine, da `IEDTEXE` einen Returncode gebracht hat. Es wird auch die Bibliothek `EDT.BEISPIELE` gefunden, die nicht mit @COPY eingelesen werden kann.

(6) Mit @STA=PAR verschafft man sich eine Übersicht über die Dateibelegung und sieht, dass offenbar in die Arbeitsdateien 1 bis 4 Dateien eingelesen wurden.

8 Anhang - C-Header

In diesem Anhang wird das Layout der C-Header-Files dargestellt. Hinweise zur Anwendung finden sich auch in den vorangehenden Abschnitten. Die äquivalenten Assembler-Makros sind in Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22 beschrieben.

8.1 Include-Dateien für die Programmierung in C

Für den Aufruf des EDT aus einem C-Programm werden Makros zur Definition, Initialisierung und Modifikation der EDT-Kontrollblöcke als Include-Dateien ausgeliefert. Die Returncodes sind als symbolische Konstanten definiert.

Es werden immer *beide* Formate (V17-Format und V16-Format) der Schnittstellen erzeugt. Diese sind stets über `strukturname_v16` bzw. `strukturname_v17` ansprechbar (z.B. `iedglcb_v17 bzw. iedglcb_v16`).

Mir dem `#define`-Wert `EDT_V17` kann der Benutzer steuern, ob die Standardbezeichnungen der Kontrollblöcke und die Kurzbezeichnungen zur Initialisierung bzw. zum Zugriff dem V17-Format oder dem V16-Format gleichgesetzt werden sollen. Ist `EDT_V17` definiert, wird das V17-Format als Standard verwendet, andernfalls das V16-Format.

Die einzelnen Kontrollblockfelder sind im Abschnitt „Generierung und Aufbau der Kontrollblöcke“ auf Seite 22 beschrieben.

8.1.1 iedtgle.h

Funktionsprototypen für die Einsprungpunkte der IEDTGLE-Schnittstelle.

Hinweis

Dieses Include setzt seinerseits ein #include für alle anderen Include-Dateien ab. Zur Verwendung der EDT-Schnittstellen genügt daher dieses Include.

```
/*
*********************************************************************
** function prototypes
*********************************************************************
*/

extern void IEDTINF(iedglcb *glcb);
extern void IEDTCMD(iedglcb *glcb,iedupcb *upcb,iedbuff *cmd,
                    iedbuff *msg1,iedbuff *msg2);
extern void IEDTEXE(iedglcb *glcb,iedbuff *cmd);
extern void IEDTGET(iedglcb *glcb,iedamcb *amcb,iedbyte *key1,
                    iedbyte *key,void *rec_or_parg_or_parl);
extern void IEDTGTM(iedglcb *glcb,iedamcb *amcb,iedbyte *key1,
                    iedbyte *key,iedbyte *rec);
extern void IEDTPUT(iedglcb *glcb,iedamcb *amcb,iedbyte *key,iedbyte *rec);
extern void IEDTPTM(iedglcb *glcb,iedamcb *amcb,iedbyte *key);
extern void IEDTDEL(iedglcb *glcb,iedamcb *amcb,iedbyte *key1,iedbyte *key2);
extern void IEDTREN(iedglcb *glcbiedamcb *amcb,iedbyte *key1,iedbyte *key2);
```

8.1.2 iedglcb.h

Definitionen und Makros für den globalen Kontrollblock `EDTGLCB` und Definition von symbolischen Konstanten für die Returncodes:

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/

#ifndef IEDT_TYPES

typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;

#define IEDT_TYPES
#endif

/*
*********************************************************************
** IEDGLCB parameter block V16
*********************************************************************
*/

typedef struct IEDGLCB_v16_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 1  */

    /* returncode structure */
    union {
        struct {
            struct {
                iedbyte subcode2;
                iedbyte subcode1;
            } subcode;
            union {
                iedshort maincode;
                struct {
                    iedbyte maincode2;
                    iedbyte maincode1;
```

```
            } main_returncode;
        } mc;
    } structured_rc;
    iedlong rc_nbr;              /* general return code */
} rc;

/* info size or displacement of invalid command */
union {
    iedlong memo_size;           /* information of memory size */
    iedlong displ_to_cmd;        /* displacement of invalid command */
} size_or_displacement;

/* return message field */
union {
    struct {
        iedshort rmsgl;          /* message length */
        iedbyte rmsgf[80];       /* message field */
    } structured_msg;
    iedbyte rmsg[82];            /* return message */
} return_message;

/* code of sending key */
iedbyte key_code;

/* indicator byte */
struct {
    iedbyte not_used_1:1;        /* not used */
    iedbyte not_used_2:1;        /* not used */
    iedbyte reorg_allowed:1;     /* reorganisation allowed */
    iedbyte not_used_3:1;        /* not used */
    iedbyte stxit_allowed:1;     /* EDT STXIT allowed */
    iedbyte data_initiated:1;    /* EDT data initiated */
    iedbyte data_add_valid:1;    /* EDT data addr. valid */
    iedbyte entry_add_valid:1;   /* EDT entry addr. valid */
} indicator;

/* EDT entry address */
void *EDT_entry;

/* EDT data address */
void *EDT_data;

/* name of actual workfile */
iedbyte filename[8];
```

```
    /* user parameter 1 */
    union {
        iedbyte user_param1_char[4];
        void *user_param1_pointer;
    } user_param1;

    /* user parameter 2 */
    union {
        iedbyte user_param2_char[4];
        void *user_param2_pointer;
    } user_param2;

    /* user parameter 3 */
    union {
        iedbyte user_param3_char[4];
        void *user_param3_pointer;
    } user_param3;

} iedglcb_v16;

/*
*********************************************************************
** IEDGLCB parameter block V17
*********************************************************************
*/

typedef struct IEDGLCB_v17_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 2  */

    /* returncode structure */
    union {
        struct {
            struct {
                iedbyte subcode2;
                iedbyte subcode1;
            } subcode;
            union {
                iedshort maincode;
                struct {
                    iedbyte maincode2;
                    iedbyte maincode1;
                } main_returncode;
            } mc;
```

```
    } structured_rc;
    iedlong rc_nbr;           /* general return code */
} rc;

/* info size or displacement of invalid command */
union {
    iedlong memo_size;        /* information of memory size */
    iedlong displ_to_cmd;     /* displacement of invalid command */
} size_or_displacement;

/* return message field */
union {
    struct {
        iedshort rmsgl;            /* message length */
        iedbyte rmsgf[80];         /* message field */
    } structured_msg;
    iedbyte rmsg[82];              /* return message */
} return_message;

/* code of sending key */
iedbyte key_code;

/* indicator byte */
struct {
    iedbyte compatible_format:1;   /* compatible/extended format */
    iedbyte ilcs_environment:1;    /* ILCS environment */
    iedbyte not_used_1:1;          /* not used */
    iedbyte not_used_2:1;          /* not used */
    iedbyte stxit_allowed:1;       /* EDT STXIT allowed */
    iedbyte data_initiated:1;      /* EDT data initiated */
    iedbyte data_add_valid:1;      /* EDT data addr. valid */
    iedbyte entry_add_valid:1;     /* EDT entry addr. valid */
} indicator;

/* EDT entry address */
void *EDT_entry;

/* EDT data address */
void *EDT_data;

/* name of actual workfile */
iedbyte filename[8];

/* user parameter 1 */
union {
    iedbyte user_param1_char[4];
    void *user_param1_pointer;
} user_param1;
```

```
    /* user parameter 2 */
    union {
        iedbyte user_param2_char[4];
        void *user_param2_pointer;
    } user_param2;

    /* user parameter 3 */
    union {
        iedbyte user_param3_char[4];
        void *user_param3_pointer;
    } user_param3;

    /* coded character set name for sub programm communication */
    iedbyte ccsn[8];

    /* indicator byte */
    struct {
        iedbyte comp_mode_running:1;   /* compatible/extended format */
        iedbyte not_used_3:1;          /* not used */
        iedbyte not_used_4:1;          /* not used */
        iedbyte not_used_5:1;          /* not used */
        iedbyte not_used_6:1;          /* not used */
        iedbyte not_used_7:1;          /* not used */
        iedbyte not_used_8:1;          /* not used */
        iedbyte not_used_9:1;          /* not used */
    } indicator2;
    iedbyte reserve[3];                /* reserved */

} iedglcb_v17;

/*
*********************************************************************
** IEDGLCB parameter block default
*********************************************************************
*/

#ifdef EDT_V17
typedef iedglcb_v17 iedglcb;
#define IEDGLCB_mdl IEDGLCB_v17_mdl
#else
typedef iedglcb_v16 iedglcb;
#define IEDGLCB_mdl IEDGLCB_v16_mdl
#endif
```

```
/*
*********************************************************************
** special values in MAINCODE
*********************************************************************
*/

/* EDT call */
#define IEDGLCBcmd_no_error          0 /* successful processing        */
#define IEDGLCBcmd_syntax_error      8 /* syntax error in command      */
#define IEDGLCBcmd_runtime_error    12 /* runtime error in command     */
#define IEDGLCBcmd_unrec_edt_error  16 /* unrecoverable EDT error      */
#define IEDGLCBcmd_unrec_sys_error  20 /* unrecoverable system error   */
#define IEDGLCBcmd_unrec_user_error 24 /* unrecoverable user error     */
#define IEDGLCBcmd_parameter_error  32 /* parameter error              */
#define IEDGLCBcmd_reqm_error       36 /* not enough space available   */
#define IEDGLCBcmd_version_error    40 /* version error                */
#define IEDGLCBcmd_abnormal_error   44 /* abnormal halt by user        */
#define IEDGLCBcmd_compatibility    48 /* V17: compatibility violation */

/* EDT access method */
#define IEDGLCBacc_no_error          0 /* successful processing        */
#define IEDGLCBacc_access_error      4 /* access error                 */
#define IEDGLCBacc_unrec_edt_error  16 /* unrecoverable EDT error      */
#define IEDGLCBacc_unrec_sys_error  20 /* unrecoverable system error   */
#define IEDGLCBacc_unrec_user_error 24 /* unrecoverable user error     */
#define IEDGLCBacc_parameter_error  32 /* parameter error              */
#define IEDGLCBacc_reqm_error       36 /* not enough space available   */

/*
*********************************************************************
** error classes in SUBCODE1
*********************************************************************
*/

/* MAINCODE: IEDGLCBcmd_no_error */
#define IEDGLCBno_error              0 /* successful processing        */
#define IEDGLCBhalt                  4 /* halt entered                 */
#define IEDGLCBhalt_text             8 /* halt with text entered       */
#define IEDGLCBreturn               12 /* return entered               */
#define IEDGLCBreturn_text          16 /* return with text entered     */
#define IEDGLCBk1_key               20 /* return with k1               */
#define IEDGLCBignore_command       24 /* only in stmnt filter:        */
                                       /* statement to be ignore       */

/* MAINCODE: IEDGLCBcmd_parameter_error */
#define IEDGLCBglcb_error            4 /* error in EDTGLCB             */
#define IEDGLCBupcb_error            8 /* error in EDTUPCB             */
#define IEDGLCBparameter_error      12 /* error in command parameter   */
```

```
#define IEDGLCBmessage_error        16 /* error in message parameter   */
#define IEDGLCBccsn_error           20 /* V17: error in ccsn parameter */
#define IEDGLCBconversion_error     24 /* V17: conversion error        */

/* MAINCODE: IEDGLCBcmd_version_error */
#define IEDGLCBstandard_version      0 /* standard version returned    */
#define IEDGLCBno_version_returned   4 /* no version returned          */

/* MAINCODE: IEDGLCBacc_no_error */
#define IEDGLCBacc_ok                0 /* no errror                    */
#define IEDGLCBnext_record           4 /* next record returned         */
#define IEDGLCBfirst_record          8 /* first record returned        */
#define IEDGLCBlast_record          12 /* last record returned         */
#define IEDGLCBfile_cleared         16 /* file cleard                  */
#define IEDGLCBcopy_buffer_cleared  20 /* copy buffer cleard           */

/* MAINCODE: IEDGLCBacc_access_error */
#define IEDGLCBput_record_truncated  4 /* put record truncated         */
#define IEDGLCBkey_truncated         8 /* key truncated (move mode)    */
#define IEDGLCBrecord_truncated     12 /* rec. truncated (move mode)   */
#define IEDGLCBfile_empty           16 /* file is empty                */
#define IEDGLCBno_marks             20 /* no marks in file             */
#define IEDGLCBfile_not_opened      24 /* file not opened              */
#define IEDGLCBfile_real_opened     28 /* file real opened (no marks)  */
#define IEDGLCBmark_not_found       32 /* mark not found   (IEDTPTM)   */
#define IEDGLCBkey_error            36 /* key error        (IEDTREN)   */
#define IEDGLCBmax_line_number      40 /* maximum line number reached  */
#define IEDGLCBrenumber_inhibited   44 /* renumber is inhibited        */
#define IEDGLCBfile_active          48 /* file is active               */

/* MAINCODE: IEDGLCBacc_parameter_error */
#define IEDGLCBacc_glcb_error        4 /* error in EDTGLCB             */
#define IEDGLCBacc_amcb_error        8 /* error in EDTAMCB             */
#define IEDGLCBfilename_error       12 /* filename error               */
#define IEDGLCBacc_function_error   16 /* access function error        */
#define IEDGLCBkey_format_error     20 /* error in key format          */
#define IEDGLCBkey_length_error     24 /* error on key length          */
#define IEDGLCBrecord_length_error  28 /* error on record length       */
#define IEDGLCBmode_byte_error      32 /* error in transfer mode       */
#define IEDGLCBunit_version_error   36 /* error in version or unit     */
#define IEDGLCBacc_ccsn_error       40 /* V17: error in ccsn parameter */
#define IEDGLCBacc_conversion_error 44 /* V17: conversion error        */
```

```
/*
*********************************************************************
** special values in KEY-CODE
*********************************************************************
*/

#define IEDGLCBkey_code_DUE  102        /* DUE */
#define IEDGLCBkey_code_F1    91        /* F1 */
#define IEDGLCBkey_code_F2    92        /* F2 */
#define IEDGLCBkey_code_F3    93        /* F3 */
#define IEDGLCBkey_code_K1    83        /* K1 */

/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/

#define IEDGLCB_UNIT_66    66
#define IEDGLCB_FUNCT_0     0
#define IEDGLCB_VERS_1      1
#define IEDGLCB_VERS_2      2

#define IEDGLCB_INIT_V16 \
    { 66,0,1,{0},0,{0},0,{0},0,0,"        ",{0},{"    "},{"    "} }
#define IEDGLCB_INIT_V17 \
    { 66,0,2,{0},0,{0},0,{0,1,0,0,0,0,0,0},0,0,"        ",{0},{"    "},
{"    "},"        ",{0,0,0,0,0,0,0,0},{0,0,0} }

#ifdef EDT_V17

#define IEDGLCB_INIT        IEDGLCB_INIT_V17
#define IEDGLCB_VERS_STD    IEDGLCB_VERS_2

#else

#define IEDGLCB_INIT        IEDGLCB_INIT_V16
#define IEDGLCB_VERS_STD    IEDGLCB_VERS_1

#endif

#define IEDGLCB_MOD_VERS(p,v)   (p).version = v
#define IEDGLCB_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v

#define IEDGLCB_RC_SUBCODE2     rc.structured_rc.subcode.subcode2
#define IEDGLCB_RC_SUBCODE1     rc.structured_rc.subcode.subcode1
#define IEDGLCB_RC_MAINCODE     rc.structured_rc.mc.maincode
```

```
#define IEDGLCB_RC_MAINCODE2    rc.structured_rc.mc.main_returncode.maincode2
#define IEDGLCB_RC_MAINCODE1    rc.structured_rc.mc.main_returncode.maincode1
#define IEDGLCB_RC_NBR          rc.rc_nbr

#define IEDGLCB_MOD_RC(p,sc2,sc1,mrc) \
    (p).IEDGLCB_RC_SUBCODE2 = sc2, \
    (p).IEDGLCB_RC_SUBCODE1 = sc1, \
    (p).IEDGLCB_RC_MAINCODE = mrc

#define IEDGLCB_RC_NIL    -1
#define IEDGLCB_RC_NULL    0

#define IEDGLCB_SET_RC_NIL(p)     (p).IEDGLCB_RC_NBR = IEDGLCB_RC_NIL
#define IEDGLCB_SET_RC_NULL(p)    (p).IEDGLCB_RC_NBR = IEDGLCB_RC_NULL

#define IEDGLCB_MSG    return_message.structured_msg.rmsgf
#define IEDGLCB_MSGL   return_message.structured_msg.rmsgl

/*
*********************************************************************
** layout of buffers (command, message)
*********************************************************************
*/

typedef struct IEDBUFF_mdl {
    iedshort length;        /* length including all fields */
    iedshort unused;        /* unused field */
    iedbyte text[1];        /* up to 256 (V16) or 32768 (V17) bytes */
} iedbuff;
```

8.1.3 iedupcb.h

Definitionen und Makros für den Unterprogramm-Kontrollblock `EDTUPCB`:

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/

#ifndef IEDT_TYPES

typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;

#define IEDT_TYPES
#endif

/*
*********************************************************************
** IEDUPCB parameter block V16
*********************************************************************
*/

typedef struct IEDUPCB_v16_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 2  */

    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;

    /* inhibit flag byte */
    union {
        struct {
            iedbyte not_used_1:1;        /* reserved */
            iedbyte no_text_at_exit:1;   /* @HALT/@RET <text> */
            iedbyte no_edit_only:1;      /* @EDIT ONLY */
            iedbyte no_edit:1;           /* @EDIT */
            iedbyte no_user_prog:1;      /* @RUN, @USE */
            iedbyte no_bkpt:1;           /* @SYSTEM */
            iedbyte no_cmd:1;            /* @SYSTEM <string> */
```

```
            iedbyte no_exec:1;          /* @EXEC/@LOAD */
        } bit;
        iedbyte byte;
    } inhibit;

    /* reserve */
    iedbyte reserve[3];

} iedupcb_v16;

/*
*********************************************************************
** IEDUPCB parameter block V17
*********************************************************************
*/

typedef struct IEDUPCB_v17_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 3  */

    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;

    /* inhibit flag byte */
    union {
        struct {
            iedbyte no_mode:1;            /* @MODE */
            iedbyte no_text_at_exit:1;    /* @HALT/@RET <text> */
            iedbyte no_edit_only:1;       /* @EDIT ONLY */
            iedbyte no_edit:1;            /* @EDIT */
            iedbyte no_user_prog:1;       /* @RUN, @USE */
            iedbyte no_bkpt:1;            /* @SYSTEM */
            iedbyte no_cmd:1;             /* @SYSTEM <string> */
            iedbyte no_exec:1;            /* @EXEC/@LOAD */
        } bit;
        iedbyte byte;
    } inhibit;

    /* reserve */
    iedbyte reserve[3];

} iedupcb_v17;
```

```
/*
*********************************************************************
** IEDUPCB parameter block default
*********************************************************************
*/

#ifdef EDT_V17
typedef iedupcb_v17 iedupcb;
#define IEDUPCB_mdl IEDUPCB_v17_mdl
#else
typedef iedupcb_v16 iedupcb;
#define IEDUPCB_mdl IEDUPCB_v16_mdl
#endif

/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/

#define IEDUPCB_UNIT_66   66
#define IEDUPCB_FUNCT_0   0
#define IEDUPCB_VERS_2    2
#define IEDUPCB_VERS_3    3

#define IEDUPCB_INIT_V16 { 66,0,2 }
#define IEDUPCB_INIT_V17 { 66,0,3 }

#ifdef EDT_V17

#define IEDUPCB_INIT       IEDUPCB_INIT_V17
#define IEDUPCB_VERS_STD   IEDUPCB_VERS_3

#else

#define IEDUPCB_INIT       IEDUPCB_INIT_V16
#define IEDUPCB_VERS_STD   IEDUPCB_VERS_2

#endif

#define IEDUPCB_MOD_VERS(p,v)   (p).version = v
#define IEDUPCB_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v

#define IEDUPCB_SET_NO_INHIBIT(p)   (p).inhibit.byte = 0
```

8.1.4 iedamcb.h

Definitionen und Makros für den Satzzugriffs-Kontrollblock EDTAMCB:

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/

#ifndef IEDT_TYPES

typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;

#define IEDT_TYPES
#endif

/*
*********************************************************************
** IEDAMCB parameter block V16
*********************************************************************
*/

typedef struct IEDAMCB_v16_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 1  */

    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;

    /* transfer mode flag byte */
    union {
        struct {
            iedbyte not_used_1:5;        /* not used */
            iedbyte locate:1;            /* locate mode */
            iedbyte not_used_2:2;        /* not used */
        } mode_bits;
        iedbyte mode_byte;               /* mode byte */
    } mode_flag;
```

```
/* flag byte */
union {
    struct {
        iedbyte not_used:6;          /* not used */
        iedbyte inh_set_modify:1;    /* inhibit setting modify flag */
        iedbyte ign_mark13:1;        /* ignore mark 13 */
    } flag_bits;
    iedbyte flag_byte;
} flag;

/* reserve */
iedbyte reserve2[2];

/* input parameters */
iedbyte filename[8];         /* workfile */
long displacement;           /* displacement */
iedshort length_key1;        /* length of key1 */
iedshort length_key2;        /* length of key2 */

/* input parameters (only in move mode) */
iedshort length_key_outbuffer;   /* length of key output buffer */
iedshort length_rec_outbuffer;   /* length of rec output buffer */

/* input/output parameters */
iedshort length_key;             /* length of key */
iedshort length_rec;             /* length of record */

/* marks */
union {
    iedshort mark_field;
    struct {
        union {
            iedbyte mark2;               /* upper marks */
            struct {
                iedbyte mark_15:1;       /* mark 15 */
                iedbyte mark_14:1;       /* mark 14 */
                iedbyte mark_13:1;       /* mark 13 */
                iedbyte nouse_1:1;       /* not used */
                iedbyte nouse_2:1;       /* not used */
                iedbyte nouse_3:1;       /* not used */
                iedbyte mark_9:1;        /* mark 9 */
                iedbyte mark_8:1;        /* mark 8 */
            } mark2_bits;
        } upper_marks;
        union {
            iedbyte mark1;               /* lower marks */
            struct {
                iedbyte mark_7:1;        /* mark 7 */
```

```
                    iedbyte mark_6:1;        /* mark 6 */
                    iedbyte mark_5:1;        /* mark 5 */
                    iedbyte mark_4:1;        /* mark 4 */
                    iedbyte mark_3:1;        /* mark 3 */
                    iedbyte mark_2:1;        /* mark 2 */
                    iedbyte mark_1:1;        /* mark 1 */
                    iedbyte nouse_4:1;       /* not used */
                } mark1_bits;
            } lower_marks;
        } mark_bytes;
    } marks;

    /* reserve */
    iedbyte reserve[2];

} iedamcb_v16;

/*
*********************************************************************
** IEDAMCB parameter block V17
*********************************************************************
*/

typedef struct IEDAMCB_v17_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 2  */

    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;

    /* not used */
    iedbyte reserve1;

    /* flag byte */
    union {
        struct {
            iedbyte not_used:6;        /* not used */
            iedbyte inh_set_modify:1;  /* inhibit setting modify flag */
            iedbyte ign_mark13:1;      /* ignore mark 13 */
        } flag_bits;
        iedbyte flag_byte;
    } flag;
```

```
/* reserve */
iedbyte reserve2[2];

/* input parameters */
iedbyte filename[8];         /* workfile */
long displacement;           /* displacement */
iedshort length_key1;        /* length of key1 */
iedshort length_key2;        /* length of key2 */

/* input parameters */
iedshort length_key_outbuffer;  /* length of key output buffer */
iedshort length_rec_outbuffer;  /* length of rec output buffer */

/* input/output parameters */
iedshort length_key;         /* length of key */
iedshort length_rec;         /* length of record */

/* marks */
union {
    iedshort mark_field;
    struct {
        union {
            iedbyte mark2;               /* upper marks */
            struct {
                iedbyte mark_15:1;       /* mark 15 */
                iedbyte mark_14:1;       /* mark 14 */
                iedbyte mark_13:1;       /* mark 13 */
                iedbyte nouse_1:1;       /* not used */
                iedbyte nouse_2:1;       /* not used */
                iedbyte nouse_3:1;       /* not used */
                iedbyte mark_9:1;        /* mark 9 */
                iedbyte mark_8:1;        /* mark 8 */
            } mark2_bits;
        } upper_marks;
        union {
            iedbyte mark1;               /* lower marks */
            struct {
                iedbyte mark_7:1;        /* mark 7 */
                iedbyte mark_6:1;        /* mark 6 */
                iedbyte mark_5:1;        /* mark 5 */
                iedbyte mark_4:1;        /* mark 4 */
                iedbyte mark_3:1;        /* mark 3 */
                iedbyte mark_2:1;        /* mark 2 */
                iedbyte mark_1:1;        /* mark 1 */
                iedbyte nouse_4:1;       /* not used */
            } mark1_bits;
        } lower_marks;
    } mark_bytes;
```

```
    } marks;

    /* reserve */
    iedbyte reserve[2];

} iedamcb_v17;

/*
*********************************************************************
** IEDAMCB parameter block default
*********************************************************************
*/

#ifdef EDT_V17
typedef iedamcb_v17 iedamcb;
#define IEDAMCB_mdl IEDAMCB_v17_mdl
#else
typedef iedamcb_v16 iedamcb;
#define IEDAMCB_mdl IEDAMCB_v16_mdl
#endif

/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/

#define IEDAMCB_UNIT_66   66
#define IEDAMCB_FUNCT_0   0
#define IEDAMCB_VERS_1    1
#define IEDAMCB_VERS_2    2

#define IEDAMCB_INIT_V16 { 66,0,1,0,{0},{0},{0},"        ",0,8,8,8,0,8, }
#define IEDAMCB_INIT_V17 { 66,0,2,0,0,{0},{0},"        ",0,8,8,8,0,8, }

#ifdef EDT_V17

#define IEDAMCB_INIT       IEDAMCB_INIT_V17
#define IEDAMCB_VERS_STD   IEDAMCB_VERS_2

#else

#define IEDAMCB_INIT       IEDAMCB_INIT_V16
#define IEDAMCB_VERS_STD   IEDAMCB_VERS_1

#endif
```

```
#define IEDAMCB_MOD_VERS(p,v)   (p).version = v
#define IEDAMCB_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v

#define IEDAMCB_SET_NO_MARKS(p)   (p).marks.mark_field = 0

#define MARK_1(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_1
#define MARK_2(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_2
#define MARK_3(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_3
#define MARK_4(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_4
#define MARK_5(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_5
#define MARK_6(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_6
#define MARK_7(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_7
#define MARK_8(p)   (p).marks.mark_bytes.upper_marks.mark2_bits.mark_8
#define MARK_9(p)   (p).marks.mark_bytes.upper_marks.mark2_bits.mark_9
#define MARK_13(p)  (p).marks.mark_bytes.upper_marks.mark2_bits.mark_13
#define MARK_14(p)  (p).marks.mark_bytes.upper_marks.mark2_bits.mark_14
#define MARK_15(p)  (p).marks.mark_bytes.upper_marks.mark2_bits.mark_15
```

8.1.5 iedparg.h

Definitionen und Makros für den Kontrollblock EDTPARG (globale Einstellungen):

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/

#ifndef IEDT_TYPES

typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;

#define IEDT_TYPES
#endif

/*
*********************************************************************
** IEDPARG parameter block V16
*********************************************************************
*/

typedef struct IEDPARG_v16_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;                  /* function unit number : 66 */
    iedbyte function;               /* function number      : 0  */
    iedbyte version;                /* interface version    : 1  */

    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;

    /* output fields */
    iedbyte EDT_mode;               /* edt modus */
    iedbyte command_symbol;         /* actual '@' */
    iedshort size_window1;          /* size of window 1 */
    iedshort size_window2;          /* size of window 2 */
    iedbyte file_in_window1[8];     /* workfile in window 1 */
    iedbyte file_in_window2[8];     /* workfile in window 2 */
    iedbyte ccs_name[8];            /* global coded character set */

} iedparg_v16;
```

```
/*
*********************************************************************
** IEDPARG parameter block V17
*********************************************************************
*/

typedef struct IEDPARG_v17_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;                /* function unit number : 66 */
    iedbyte function;             /* function number      : 0  */
    iedbyte version;              /* interface version    : 2  */

    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;

    /* output fields */
    iedbyte EDT_mode;             /* edt modus */
    iedbyte command_symbol;       /* actual '@' */
    iedshort size_window1;        /* size of window 1 */
    iedshort size_window2;        /* size of window 2 */
    iedbyte file_in_window1[8];   /* workfile in window 1 */
    iedbyte file_in_window2[8];   /* workfile in window 2 */

    /* reserve */
    iedbyte reserve[10];

} iedparg_v17;

/*
*********************************************************************
** IEDPARG parameter block default
*********************************************************************
*/

#ifdef EDT_V17
typedef iedparg_v17 iedparg;
#define IEDPARG_mdl IEDPARG_v17_mdl
#else
typedef iedparg_v16 iedparg;
#define IEDPARG_mdl IEDPARG_v16_mdl
#endif
```

```
/*
*********************************************************************
** special values in EDT_mode
*********************************************************************
*/

#define IEDPARGmode_fullscreen  0xC6        /* 'F' full screen mode */
#define IEDPARGmode_line        0xD3        /* 'L' line mode */
#define IEDPARGmode_control     0xC3        /* 'C' user control */

/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/

#define IEDPARG_UNIT_66    66
#define IEDPARG_FUNCT_0     0
#define IEDPARG_VERS_1      1
#define IEDPARG_VERS_2      2

#define IEDPARG_INIT_V16 \
    { 66,0,1,0,0,0,0,0," "," "," " }
#define IEDPARG_INIT_V17 \
    { 66,0,2,0,0,0,0,0," "," " }

#ifdef EDT_V17

#define IEDPARG_INIT       IEDPARG_INIT_V17
#define IEDPARG_VERS_STD   IEDPARG_VERS_2

#else

#define IEDPARG_INIT       IEDPARG_INIT_V16
#define IEDPARG_VERS_STD   IEDPARG_VERS_1

#endif

#define IEDPARG_MOD_VERS(p,v)    (p).version = v
#define IEDPARG_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v
```

8.1.6 iedparl.h

Definitionen und Makros für den Kontrollblock EDTPARL (arbeitsdateispezifische Einstellungen):

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/

#ifndef IEDT_TYPES

typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;

#define IEDT_TYPES
#endif

/*
*********************************************************************
** IEDPARL parameter block V16
*********************************************************************
*/

typedef struct IEDPARL_v16_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 3  */

    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;

    /* output fields */
    iedbyte first_line_window[8]; /* number of first line */

    /* in window */
    iedshort first_col_window;    /* first column in window */
    iedshort record_length_max;   /* max. record length in */
```

```
    /* in fullscreen mode */
    iedbyte par_inf;          /* INF on/off (1/0) */
    iedbyte par_low;          /* LOWER on/off (1/0) */
    iedbyte par_hex;          /* HEX on/off (1/0) */
    iedbyte par_edit_long;    /* EDIT-LONG on/off (1/0) */
    iedbyte par_scale;        /* SCALE on/off (1/0) */
    iedbyte par_protection;   /* PROTECTION on/off (1/0) */
    iedbyte structure_symbol; /* structure symbol */
    iedbyte open_flag;        /* open flag (I/P/R/S/X/0) */
    iedbyte empty_flag;       /* empty y/n (1/0) */
    iedbyte modified_flag;    /* modified y/n (1/0) */
    iedbyte std_file[54];     /* standard file name */
    iedbyte std_library[54];  /* standard library name */
    iedbyte std_plam_type[8]; /* standard plam type */
    iedbyte std_code;         /* standard code (E/I) */
    iedbyte not_used1[3];     /* reserved */
    iedbyte first_line1[8];   /* number of first line in window 1 */
    iedshort first_col1;      /* first column in window 1 */
    iedbyte first_line2[8];   /* number of first line in window 2 */
    iedshort first_col2;      /* first column in window 2 */
    iedbyte index_window1;    /* INDEX OFF/ON/FULL (0/1/2) window 1 */
    iedbyte index_window2;    /* INDEX OFF/ON/FULL (0/1/2) window 2 */

    /* description of opened data file */
    union {
        iedbyte common_area[260];     /* common area */
        struct {
            iedbyte file_name[54];    /* name of opened file or lib */
            iedbyte plam_elem[64];    /* name of plam element */
            iedbyte plam_vers[24];    /* name of plam version */
            iedbyte plam_type[8];     /* plam type */
        } file_or_plam_elem;
        struct {
            iedbyte ufs_name[256];    /* name of opened ufs file */
            iedbyte code;             /* code of opened ufs file (?)*/
        } ufs_file;
    } file_description;

    /* reserved */
    iedbyte not_used2[8];

} iedparl_v16;
```

```
/*
*********************************************************************
** IEDPARL parameter block V17
*********************************************************************
*/

typedef struct IEDPARL_v17_mdl {

    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;            /* function unit number : 66 */
    iedbyte function;         /* function number      : 0  */
    iedbyte version;          /* interface version    : 4  */

    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;

    /* output fields */
    iedbyte first_line_window[8]; /* number of first line */

    /* in window */
    iedshort first_col_window;  /* first column in window */
    iedshort record_length_max; /* max. record length in */

    /* in fullscreen mode */
    iedbyte par_inf;          /* INF on/off (1/0) */
    iedbyte par_low;          /* LOWER on/off (1/0) */
    iedbyte par_hex;          /* HEX on/off (1/0) */
    iedbyte par_edit_long;    /* EDIT-LONG on/off (1/0) */
    iedbyte par_scale;        /* SCALE on/off (1/0) */
    iedbyte par_protection;   /* PROTECTION on/off (1/0) */
    iedbyte structure_symbol; /* structure symbol (if EBCDIC) */
    iedbyte open_flag;        /* open flag (I/P/R/S/X/0) */
    iedbyte empty_flag;       /* empty y/n (1/0) */
    iedbyte modified_flag;    /* modified y/n (1/0) */
    iedbyte std_file[54];     /* standard file name */
    iedbyte std_library[54];  /* standard library name */
    iedbyte std_plam_type[8]; /* standard plam type */
    iedbyte not_used1[4];     /* reserved */
    iedbyte first_line1[8];   /* number of first line in window 1 */
    iedshort first_col1;      /* first column in window 1 */
    iedbyte first_line2[8];   /* number of first line in window 2 */
    iedshort first_col2;      /* first column in window 2 */
    iedbyte index_window1;    /* INDEX OFF/ON/FULL (0/1/2) window 1 */
    iedbyte index_window2;    /* INDEX OFF/ON/FULL (0/1/2) window 2 */
```

```
    /* description of opened data file */
    union {
        iedbyte common_area[1024];  /* common area */
        struct {
            iedbyte file_name[54];  /* name of opened file or lib */
            iedbyte plam_elem[64];  /* name of plam element */
            iedbyte plam_vers[24];  /* name of plam version */
            iedbyte plam_type[8];   /* plam type */
        } file_or_plam_elem;
        struct {
            iedbyte ufs_name[1024]; /* name of opened ufs file */
        } ufs_file;
    } file_description;

    /* charset information */
    iedbyte ccsn[8];                /* coded character set name */
    iedbyte ccsn_global;            /* ccsn is global (1/0) */

    /* TODO: other local par settings? */
    iedutf16 sym_structure;         /* structure symbol */

} iedparl_v17;

/*
*********************************************************************
** IEDPARL parameter block default
*********************************************************************
*/

#ifdef EDT_V17
typedef iedparl_v17 iedparl;
#define IEDPARL_mdl IEDPARL_v17_mdl
#else
typedef iedparl_v16 iedparl;
#define IEDPARL_mdl IEDPARL_v16_mdl
#endif
```

```
/*
*********************************************************************
** special values in open_flag
*********************************************************************
*/

#define IEDPARLopen_isam    0xC9 /* 'I' ISAM file virtually opened */
#define IEDPARLopen_plam    0xD7 /* 'P' PLAM element opened */
#define IEDPARLopen_real    0xD9 /* 'R' ISAM file really opened */
#define IEDPARLopen_sam     0xE2 /* 'S' SAM file virtually opened */
#define IEDPARLopen_ufs     0xE7 /* 'X' UFS file virtually opened */
#define IEDPARLopen_no      0xD6 /* 'O' no file opened */

/*
*********************************************************************
** special on/off values
*********************************************************************
*/

#define IEDPARLoff           0xF0 /* '0' OFF */
#define IEDPARLon            0xF1 /* '1' ON */

/*
*********************************************************************
** special values in index
*********************************************************************
*/

#define IEDPARLindex_off     0xF0 /* '0' INDEX OFF */
#define IEDPARLindex_on      0xF1 /* '1' INDEX ON */
#define IEDPARLindex_full    0xF2 /* '2' EDIT FULL */

/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/

#define IEDPARL_UNIT_66    66
#define IEDPARL_FUNCT_0     0
#define IEDPARL_VERS_3      3
#define IEDPARL_VERS_4      4

#define IEDPARL_INIT_V16 { 66,0,3 }
#define IEDPARL_INIT_V17 { 66,0,4 }

#ifdef EDT_V17
```

```
#define IEDPARL_INIT        IEDPARL_INIT_V17
#define IEDPARL_VERS_STD    IEDPARL_VERS_4

#else

#define IEDPARL_INIT        IEDPARL_INIT_V16
#define IEDPARL_VERS_STD    IEDPARL_VERS_3

#endif

#define IEDPARL_MOD_VERS(p,v)    (p).version = v
#define IEDPARL_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v
```

Fachwörter

@DO-Prozedur

Eine @DO-Prozedur ist eine EDT-Prozedur, die in einer Arbeitsdatei gespeichert ist. Sie kann mit der Anweisung @DO zum Ablauf gebracht werden. In @DO-Prozeduren stehen einige Anweisungen zur Ablaufsteuerung zur Verfügung. Bei ihrem Aufruf können Parameter übergeben werden und sie können geschachtelt werden.

@INPUT-Prozedur

Eine @INPUT-Prozedur ist eine EDT-Prozedur, die in einer Datei oder in einem Bibliothekselement gespeichert ist. Sie kann mit der Anweisung @INPUT zum Ablauf gebracht werden. In @INPUT-Prozeduren stehen die Anweisungen zur Ablaufsteuerung nicht (direkt) zur Verfügung, sie können nicht ineinandergeschachtelt werden und es können keine Parameter übergeben werden. Es können aber @DO-Prozeduren aufgerufen werden.

Anweisung

Eingaben an den EDT sind entweder Datensätze oder Anweisungen. Mit Anweisungen können Funktionen des EDT ausgelöst bzw. Einstellungen vorgenommen werden. Um Anweisungen von Datensätzen unterscheiden zu können, müssen Anweisungen im Line-Modus mit dem EDT-Anweisungssymbol eingeleitet werden. Im F-Modus werden Anweisungen in der Anweisungszeile eingegeben. Daneben gibt es noch die Kurzanweisungen, die in der Kurzanweisungsspalte eingegeben werden.

Anweisungspuffer

Der EDT speichert die letzten im F-Modus eingegebenen Anweisungen in einem Puffer, aus dem diese wieder zurückgeholt werden können.

Anweisungszeile

Ein Feld des Arbeitsfensters im F-Modus. Eingaben in der Anweisungszeile werden als Anweisungen interpretiert. Das EDT-Anweisungssymbol kann normalerweise weggelassen werden.

Arbeitsdatei

Eingabe und Bearbeitung von Daten geschieht im EDT immer in einer Arbeitsdatei. In Arbeitsdateien können z.B. Daten eingefügt, geändert und gelöscht werden. Der Inhalt der Arbeitsdateien kann am Bildschirm angezeigt werden. Sollen Inhalte einer Datei (DVS-Datei, Bibliothekselement oder POSIX-Datei) bearbeitet werden, müssen sie zunächst in eine Arbeitsdatei übernommen werden. Nach der Bearbeitung kann der Inhalt einer Arbeitsdatei in eine Datei geschrieben werden.
Der EDT kann 23 Arbeitsdateien verwalten. Die Arbeitsdateien sind in Sätzen organisiert, denen eine Zeilennummer zugeordnet ist.

Arbeitsdatei, aktuelle

Eine Arbeitsdatei ist die aktuelle Arbeitsdatei. In ihr werden Daten eingegeben und Anweisungen wirksam. Im F-Modus wird ein Ausschnitt der aktuellen Arbeitsdatei am Bildschirm angezeigt.

Arbeitsdatei, aktive

Eine **aktive** Arbeitsdatei ist eine Arbeitsdatei, die eine @DO-Prozedur enthält, die gerade ausgeführt wird. Wenn @DO-Prozeduren geschachtelt werden, können mehrere Arbeitsdateien aktiv sein.

Arbeitsdatei, leere

Eine **leere** Arbeitsdatei ist eine Arbeitsdatei, die keine Sätze enthält. Auch eine leere Arbeitsdatei kann noch Eigenschaften besitzen, die nicht dem Initialzustand entsprechen, z.B. kann sie noch als belegt gelten oder mit einer Datei verknüpft sein. Erst mit einer Anweisung @DELETE (Format 2) oder mit anderen Anweisungen, die implizit oder explizit Arbeitsdateien vollständig löschen, wird die Arbeitsdatei wieder in den Initialzustand versetzt.

Arbeitsfenster

Im F-Modus wird die aktuelle Arbeitsdatei am Bildschirm dargestellt. Der Bildschirm wird dabei in Felder mit unterschiedlicher Funktion aufgeteilt. Neben dem Datenfenster, in dem der Inhalt der aktuellen Arbeitsdatei dargestellt wird, enthält das Arbeitsfenster u.a. noch eine Anweisungszeile und eine Kurzanweisungsspalte.

Arbeitsmodus

Im EDT stehen zwei Arbeitsmodi für die Bearbeitung von Daten zur Verfügung, der Line-Modus (L-Modus) und der Full-Screen-Modus (F-Modus).
Im L-Modus wird jeweils nur eine Bildschirmzeile zur Eingabe von Daten und Anweisungen angeboten.
Im F-Modus steht der gesamte Bildschirm für die Eingabe von Daten und Anweisungen zur Verfügung.
Die Arbeitsmodi sind nicht zu verwechseln mit den Betriebsmodi des EDT (Kompatibilitäts-Modus und Unicode-Modus). Während die Arbeitsmodi sich auf die Unterschiede bei der Darstellung und der Arbeit mit den Daten beziehen, stellen die Betriebsmodi unterschiedliche EDT-Umgebungen mit eingeschränktem bzw. erweitertem Funktionsumfang dar.

Begrenzersymbole

Literale werden normalerweise durch das Begrenzersymbol `apostrophe` (Standardwert ') eingeschlossen. Beim Suchen mit der Anweisung @ON kann auch ein besonderes Begrenzersymbol verwendet werden, das `quotation mark` (Standardwert "). Damit wird festgelegt, dass eine Zeichenfolge nur dann als Treffer erkannt wird, wenn sie durch Textbegrenzerzeichen begrenzt ist. Die Textbegrenzerzeichen sind eine einstellbare Menge von Zeichen, zu denen Standardmäßig das Leerzeichen, die runden Klammern u.a. gehören.

Benutzeranweisungssymbol

Ein Benutzeranweisungssymbol ist ein spezielles Zeichen, mit dem Benutzeranweisungen gekennzeichnet werden, die durch externe Anweisungsroutinen ausgeführt werden.

Betriebsarten

Es werden zwei Betriebsarten unterschieden, je nach dem ob eine Datensichtstation vorhanden ist oder nicht. Im Dialogbetrieb ist eine Datensichtstation vorhanden, im Stapelbetrieb nicht.

Betriebsmodus

Da die notwendigen Erweiterungen für die Unicode-Unterstützung des EDT nicht kompatibel erfolgen konnten, wurde ein neuer Betriebsmodus des EDT eingeführt. EDT V17.0A kann also in 2 Modi betrieben werden, dem Unicode-Modus und dem Kompatibilitäts-Modus.
Im Unicode-Modus steht eine Reihe von Erweiterungen zur Verfügung. Vor allem können (nur) im Unicode-Modus Unicode-Dateien bearbeitet werden. Allerdings ist dieser Modus nicht in allen Punkten kompatibel zu EDT V16.6B. Der Kompatibilitäts-Modus gewährleistet dagegen die volle Funktionalität des EDT V16.6B, dafür stehen allerdings die Erweiterungen nicht zur Verfügung. EDT V17.0A wird standardmäßig im Kompatibilitäts-Modus gestartet. Mit einer neuen Anweisung kann in den Unicode-Modus umgeschaltet werden.

Bildschirmdialog

Mit der Anweisung @DIALOG, die nur an der Unterprogrammschnittstelle bzw. von SYSDTA eingegeben werden kann, kann der EDT in den Bildschirmdialog versetzt werden. Im Bildschirmdialog ist der bisherige Lesevorgang unterbrochen und der EDT liest seine Eingaben im F-Modus (oder im L-Modus nach Eingabe von @EDIT) von der Datensichtstation. Der Bildschirmdialog kann mit @HALT, @END, @RETURN oder K1 wieder verlassen werden. Der EDT setzt dann den unterbrochenen Lesevorgang fort.

Bildschirmzeile

Zeilen des Datenfensters, in denen die Sätze der aktuellen Arbeitsdatei angezeigt werden.

Datenfenster

Feld des Arbeitsfensters, in dem die aktuelle Arbeitsdatei angezeigt wird. Die Sätze der Arbeitsdatei werden in die Bildschirmzeilen des Datenfensters ausgegeben.

Dialogbetrieb

Der Dialogbetrieb ist die Betriebsart des EDT, bei der eine Datensichtstation vorhanden ist. Nur in dieser Betriebsart kann der EDT im F-Modus arbeiten.

EDT-Anweisungssymbol

Das EDT-Anweisungssymbol ist das spezielle Zeichen, mit dem Anweisungen gekennzeichnet werden. Standardmäßig ist dies das Zeichen @, das daher in diesem Dokument immer verwendet wird.

EDT-Prozeduren

Eine EDT-Prozedur ist eine Folge von Eingaben, Anweisungen und/oder Datensätzen, an den EDT, die in einer Datei (@INPUT-Prozedur) bzw. einer Arbeitsdatei (@DO-Prozedur) abgelegt sind.

EDT-Startprozedur

Die EDT-Startprozedur ist eine spezielle @INPUT-Prozedur, die (falls vorhanden) beim Start des EDT ausgeführt wird.

EDT-Variablen

EDT-Variablen sind Behälter, in denen Werte auch über Arbeitsdateien hinweg gespeichert werden können. EDT-Variablen sind nur während des EDT-Laufs gültig.

Es gibt drei Arten von Variablen, die mit entsprechenden Werten versorgt werden können. Von jeder Variablenart stehen 21 Variablen zur Verfügung:

- Ganzzahlvariablen (#I0..#I20)
- Zeichenfolgevariablen (#S0..#S20)
- Zeilennummervariablen (#L0..#L20)

Ersatzzeichen

Wenn Zeichenfolgen aus einem Zeichensatz in einen anderen Zeichensatz konvertiert werden, kann der Fall eintreten, dass Zeichen aus den Quell-Daten im Ziel-Zeichensatz nicht vorhanden sind. Stattdessen wird dann das Ersatzzeichen eingesetzt, falls es definiert wurde. Wenn kein Ersatzzeichen definiert wurde, wird die Konvertierung normalerweise abgelehnt.

Full-Screen-Modus (F-Modus)

Der Full-Screen-Modus (F-Modus) ist ein Arbeitsmodus des EDT. Im F-Modus steht der gesamte Bildschirm in Form eines Arbeitsfensters für die Eingabe von Daten und Anweisungen zur Verfügung. Es besteht die Möglichkeit vom F-Modus in den L-Modus umzuschalten. Der EDT kann nur im Dialogbetrieb im F-Modus arbeiten.

Kompatibilitäts-Modus

Der Kompatibilitäts-Modus ist ein Betriebsmodus des EDT V17.0A. Im Kompatibilitäts-Modus ist die volle Funktionalität des EDT V16.6B gewährleistet. Allerdings können die erweiterten Funktionen des EDT V17.0A nicht genutzt werden z.B. erlaubt er nicht die Bearbeitung von Unicode-Dateien und die Satzlänge ist weiterhin auf 256 Byte beschränkt.
Ein Wechsel in den Unicode-Modus erfolgt unter gewissen Voraussetzungen implizit, wenn eine Unicode-Datei eingelesen wird, oder explizit, wenn eine @MODE-Anweisung eingegeben wird.

Kommunikations-Zeichensatz

Zeichensatz, in dem der EDT im Unicode-Modus mit der Datensichtstation kommuniziert. Dies kann ein anderer sein als der Zeichensatz der aktuellen und auch jeder anderen Arbeitsdatei. Der Kommunikations-Zeichensatz ist normalerweise optimal an die Möglichkeiten der Datensichtstation angepasst.

Kurzanweisung

Kurzanweisungen sind 1 Zeichen lange Anweisungen, die im F-Modus in der Kurzanweisungsspalte eingegeben werden können.

Kurzanweisungsspalte

Die Kurzanweisungsspalte ist ein Feld des Arbeitsfensters, in dem Kurzanweisungen eingegeben werden können.

Line-Modus (L-Modus)

Der Line-Modus (L-Modus) ist ein Arbeitsmodus des EDT. Im L-Modus erfolgt die Dateibearbeitung zeilenorientiert, d.h. der EDT bietet im Dialogbetrieb nur eine Zeile (die aktuelle Zeile) zur Eingabe an bzw. liest (im Stapel- und im Dialogbetrieb) jeweils eine Zeile von SYSDTA. Diese Zeile kann entweder Datensätze oder Anweisungen enthalten und wird nach dem Einlesen sofort verarbeitet.

Musterzeichen

Musterzeichen sind Platzhalter für Zeichengruppen in einem Suchstring. Dabei steht asterisk (Standardwert *) für eine beliebig lange, auch leere Zeichenfolge, slash (Standardwert /) steht für genau ein Zeichen.

Satzmarkierungen

Jeder Satz einer Arbeitsdatei kann mit Satzmarkierungen gekennzeichnet werden, die für den Benutzer nicht sichtbar sind. Sie können mit Anweisungen und Kurzanweisungen gesetzt, abgefragt und gelöscht werden. Markierte Sätze können dann z.B. kopiert oder gelöscht werden.

Stapelbetrieb

Der Stapelbetrieb ist die Betriebsart des EDT, wenn keine Datensichtstation vorhanden ist. Der EDT kann dann nur im L-Modus arbeiten.

Unicode-Ersatzdarstellung

Der EDT erlaubt in Anweisungen innerhalb von Literalen, aber auch in Daten, über eine Ersatzdarstellung, Unicode-Zeichen durch die Angabe des Hex-Wertes ihres UTF16 Codes anzugeben. Damit können über ein Fluchtsymbol alle (unterstützten) Zeichen eingegeben werden, auch wenn der Zeichensatz der Anweisung bzw. der Daten kein Unicode-Zeichensatz ist.

Unicode-Modus

Der Unicode-Modus ist ein Betriebsmodus des EDT. Nur im Unicode-Modus stehen die erweiterten Funktionen des EDT V17.0A zur Verfügung, d.h. nur in diesem Modus können Unicode-Dateien bearbeitet werden, können Sätze länger als 256 Byte sein, gibt es lokale Zeichensätze usw.
Allerdings ist dieser Modus nicht in allen Punkten kompatibel zu EDT V16.6B. Insbesondere gibt es Inkompatibilitäten an der Unterprogrammschnittstelle. Außerdem ist eine Reihe von Inkonsistenzen bereinigt worden.

Zeichensatz

Mit dem EDT V17.0A können Texte in allen Zeichensätzen bearbeitet werden, die XHCS bereitstellt. Zusätzlich zu den in EDT V16.6B bzw. im Kompatibilitätsmodus unterstützten sind dies im Unicode-Modus die 3 Unicode-Zeichensätze, ISO-Zeichensätze und ggf. 7-Bit-Zeichensätze.
In jeder Arbeitsdatei kann ein anderer Zeichensatz eingestellt werden, so dass also Texte in unterschiedlichen Zeichensätzen parallel bearbeitet werden können. Für die Kommunikation mit einer Datensichtstation wird ein Kommunikations-Zeichensatz eingestellt.

Zeilennummer

Jedem Satz einer Arbeitsdatei ist eine Zeilennummer zugeordnet, durch die er eindeutig gekennzeichnet ist. Eine Zeilennummer ist die aktuelle Zeilennummer, in die im L-Modus die Dateneingabe erfolgt.

Literatur

Die Handbücher sind online unter *http://manuals.fujitsu-siemens.com* zu finden oder in gedruckter Form gegen gesondertes Entgelt unter *http://FSC-manualshop.com* zu bestellen.

[1] **EDT V17.0A UNICODE-Modus** (BS2000/OSD)
Anweisungen
Benuterhandbuch

[2] **EDT V16.6B** (BS2000/OSD)
Anweisungen
Benutzerhandbuch

[3] **EDT V16.6** (BS2000/OSD)
Unterprogrammschnittstellen
Benutzerhandbuch

[4] **EDT-ARA** (BS2000/OSD)
Additional Information for Arabic
User Guide

[5] **EDT-FAR** (BS2000/OSD)
Additional Information for Farsi
User Guide

[6] **SDF** (BS2000/OSD)
Einführung in die Dialogschnittstelle SDF
Benutzerhandbuch

[7] **SDF-P** (BS2000/OSD)
Programmieren in der Kommandosprache
Benuterhandbuch

[8] **XHCS** (BS2000/OSD)
8-bit-Code-Verarbeitung im BS2000/OSD)
Benutzerhandbuch

[9] **JV** (BS2000/OSD)
Jobvariablen
Benuterhandbuch

[10] **BS2000/OSD-BC**
Kommandos Band 1-5
Benutzerhandbuch

[11] **BS2000/OSD-BC**
Kommandos Band 6, Ausgabe in S-Variablen und SDF-BASYS
Benutzerhandbuch

[12] **BS2000/OSD-BC**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

[13] **LMS** (BS2000/OSD)
SDF-Format
Benutzerhandbuch

[14] **POSIX** (BS2000/OSD)
Grundlagen für Anwender und Systemverwalter
Benutzerhandbuch

[15] **POSIX** (BS2000/OSD)
Kommandos
Benutzerhandbuch

[16] **ASSEMBH** (BS2000/OSD)
Beschreibung

[17] **ASSEMBH** (BS2000/OSD)
Benutzerhandbuch

[18] **Datensichtstationen** (TRANSDATA)
Codetabellen
Benutzerhandbuch

Stichwörter

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.