Deutsch

FUJITSU

COBOL85 V2.3

COBOL-Compiler

Benutzerhandbuch

Ausgabe Juli 2014

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@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2008

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 9001:2008 erfüllt.

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

Copyright und Handelsmarken

Inhalt

1 Einleitung

1.1 Konzept des Handbuchs

Dieses Benutzerhandbuch beschreibt, wie COBOL-Programme im Betriebssystem BS2000

- für die Übersetzung bereitgestellt,
- mit dem COBOL85-Compiler übersetzt,
- zu ablauffähigen Programmen gebunden und in den Hauptspeicher geladen sowie
- in Testläufen auf logische Fehler untersucht werden können.

Es gibt außerdem Aufschluß darüber, wie COBOL-Programme

- die Möglichkeiten des BS2000 zum Informationsaustausch nutzen,
- katalogisierte Dateien verarbeiten,
- sortieren und mischen,
- Fixpunkte ausgeben und für einen Wiederanlauf verwenden sowie
- mit weiteren Programmen verknüpft werden können.

Ferner beschreibt dieses Benutzerhandbuch in Kapitel 13 den Einsatz des COBOL85-Compilers und der von ihm erzeugten Programme im POSIX-Subsystem des BS2000/OSD sowie den Zugriff auf das POSIX-Dateisystem.

Der Leser benötigt Kenntnisse der Programmiersprache COBOL sowie einfacher Anwendungen des BS2000.
Der Sprachumfang des COBOL85-Compilers ist im Handbuch „COBOL85-Sprachbeschreibung“ [1] dargestellt.

Auf Druckschriften wird im Text durch Kurztitel oder Nummern in eckigen Klammern hingewiesen. Die vollständigen Titel sind unter den entsprechenden Nummern im Literaturverzeichnis aufgeführt.

1.2 Die Ausbaustufen des COBOL85-Systems

Das COBOL85-System V2.3 wird in drei Ausbaustufen geliefert:

– COBOL85-R (Vollausbau mit RISC-Codegenerator)

– COBOL85 (Vollausbau ohne RISC-Codegenerator)

– COBOL85-BC (Basic Configuration / Grundausbaustufe)

In der BC-Version des COBOL85 werden folgende Steuerungs- und Sprachkomponenten nicht unterstützt:

– Dialogtesthilfe AID

– Ausgabe einer Liste aller Fehlermeldungen

– Ausgabe von Objektlisten

– COBOL-DML-Sprachelemente für Datenbankanschluß

– Sprachmodul Report-Writer

– COBOL85-Strukturierer

– Compiler- und Programmablauf im POSIX-Subsystem

– RISC-Codegenerator

Dieses Benutzerhandbuch referiert grundsätzlich die Vollausbaustufe; die Texte zu den von COBOL85-BC nicht unterstützten Funktionen enthalten einen entsprechenden Hinweis.

Der COBOL85-Compiler wird ab der Version 2.1A ohne COBOL85-Laufzeitsystem ausgeliefert.
Das COBOL85-Laufzeitsystem ist Bestandteil des CRTE (Common RunTime Environment), der gemeinsamen Laufzeitumgebung für COBOL85-, C- und C++-Programme.
Das in CRTE enthaltene COBOL85-Laufzeitsystem unterstützt den Ablauf aller Programme, die von COBOL85-Compilern ab Version1.0A übersetzt wurden.

1.3 Änderungen gegenüber dem vorigen Handbuch

In folgender Tabelle sind die wesentlichen fachlichen Neuerungen und Änderungen unter Angabe des Abschnitts aufgeführt.
Die über das ganze Handbuch verteilten inhaltlichen und sprachlichen Korrekturen sind nicht eigens genannt.

Kapitel / Abschnitt	**Stichwort**	**neu**	**geän-dert**	**entfallen**
3.3.4	COMPILER-ACTION-Option: Erweiterung um Operanden für Segmentierung und RISC-CODE SEGMENTATION = DESTINATION-CODE=	X		
3.3.6	LISTING-Option: Erweiterung um Kennzeichnung von neuen Schlüsselwörtern nach zukünftigem Standard: MARK-NEW-KEYWORDS Arbeiten mit Jahreszahlen ohne Jahrhundert: REPORT-2-DIGIT-YEAR	X		
4.2	COMOPT-Operanden für Segmentierung, RISC-CODE, Kennzeichnung von neuen Schlüsselwörtern nach zukünftigem Standard, Generieren von schnellerem Code beim Übertragen von numerischen Datenfeldern(USAGE DISPLAY): ELABORATE-SEGMENTATION GENERATE-RISC-CODE MARK-NEW-KEYWORDS INHIBIT-BAD-SIGN-PROPAGATION	X		
5	Strukturiere,Ergänzungen zur Beschreibung des Sprachumfangs	X		
16	Liste der Compiler-Module Liste der COBOL85-Laufzeitmodule		X	X

1.4 Im Handbuch verwendete Darstellungsmittel

In diesem Benutzerhandbuch werden folgende metasprachliche Konventionen verwendet:

COMOPT	Großbuchstaben bezeichnen Schlüsselwörter, die in dieser Form eingegeben werden müssen.
name	Kleinbuchstaben bezeichnen Variablen, die bei der Eingabe durch aktuelle Werte ersetzt werden müssen.
YES NO	Die Unterstreichung eines Wertes bedeutet, daß es sich um einen Standardwert handelt, der automatisch eingesetzt wird, wenn der Anwender keine Angaben macht.
{ YES NO }	Geschweifte Klammern schließen Alternativen ein, d.h. aus den angegebenen Größen muß eine Angabe ausgewählt werden. Die Alternativen stehen untereinander. Befindet sich unter den angegebenen Größen ein Standardwert, dann ist keine Angabe erforderlich, wenn der Standardwert gewünscht ist.
{YES/NO}	Ein Schrägstrich zwischen nebeneinander stehenden Angaben bedeutet ebenfalls, daß es sich um Alternativen handelt, von denen eine ausgewählt werden muß. Falls der angegebene Standardwert gewünscht wird, ist keine Angabe erforderlich.
[ ]	Eckige Klammern schließen Wahlangaben ein, die weggelassen werden dürfen.
( )	Runde Klammern müssen mit angegeben werden.
␣	Dieses Zeichen deutet an, daß mindestens ein Leerzeichen syntaktisch notwendig ist.
Sonderzeichen	sind ohne Veränderung zu übernehmen.

Hinweise

– Für COBOL-Sprachformate gelten die üblichen COBOL-Konventionen (siehe Handbuch „COBOL85-Sprachbeschreibung“ [1]).

– Die SDF-Metasyntax ist in Abschnitt 3.2 gesondert beschrieben.

1.5 Begriffserklärungen

In der Beschreibung des Programmerstellungsprozesses werden häufig unterschiedliche Begriffe für dasselbe Objekt verwendet. Beispielsweise wird das Resultat eines Compilerlaufs als „Objektmodul“ bezeichnet, während für den Binder dasselbe Objekt ein „Bindemodul“ (= „zu bindender Modul“) ist.

Die Verwendung der komponentenspezifischen Begriffe ist sinnvoll, kann aber beim Leser des Handbuchs zur terminologischen Verunsicherung führen. Um dem vorzubeugen, sind nachfolgend die wichtigsten synonym verwendeten Begriffe erklärt.

Bindemodul, Objektmodul, Großmodul

Der Begriff „Bindemodul“ faßt die beiden Begriffe „Objektmodul“ und „Großmodul“ zusammen.

Objektmodule und Großmodule sind gleichartig aufgebaut und werden im gleichen Format abgelegt (Objektmodulformat). In PLAM-Bibliotheken sind sie Elemente vom Typ R.

Objektmodule erzeugt der Compiler bei der Übersetzung von Quellprogrammen.

Großmodule, auch „vorgebundene Module“ genannt, erzeugt der Binder TSOSLNK.
In einem Großmodul sind mehrere Objekt- bzw. Großmodule in einem einzigen Modul zusammengefaßt.

Bindemodule können vom statischen Binder TSOSLNK, vom dynamischen Bindelader DBL oder vom Binder BINDER weiterverarbeitet werden.

Modul, Objektmodul, Bindelademodul

„Modul“ ist der Oberbegriff für das Ergebnis der Übersetzung eines Quellprogramms durch den COBOL85-Compiler. „Objektmodul“ ist ein Modul im OM-Format, „Bindelademodul“ ist ein Modul im LLM-Format.

Ablauffähiges Programm, Programm, Lademodul, Objektprogramm

Ein ablauffähiges Programm, in diesem Handbuch auch kurz „Programm“ genannt, wird von den Bindern erzeugt und z.B. in PLAM-Bibliotheken unter dem Typ C abgelegt.
Im Unterschied zu Bindemodulen können ablauffähige Programme nicht vom Binder TSOSLNK weiterverarbeitet werden, sondern werden vom (statischen) Lader in den Speicher geladen.

In anderer Dokumentation wird für ablauffähige Programme oft synonym der Begriff „Lademodul“ verwendet. Technisch gesehen ist jedoch ein Lademodul eine ladbare Einheit **innerhalb** eines Programms. Ein segmentiertes Programm besteht z.B. aus mehreren Lademodulen.

Das Synonym „Objektprogramm“ für Lademodul kann in der COBOL-Terminologie zu Mißverständnissen führen: Im COBOL-Standard wird, ohne auf die herstellerspezifische Notwendigkeit eines Bindelaufs einzugehen, als Objektprogramm bereits das vom COBOL-Compiler erzeugte Objekt bezeichnet.

Auftrag (Job), Task, Prozeß

Ein Auftrag (Job) ist die Folge von Kommandos, Anweisungen etc., die zwischen den Kommandos LOGON und LOGOFF angegeben werden. Es wird zwischen Stapelaufträgen (ENTER-Jobs) und Dialogaufträgen unterschieden.

Ein Auftrag wird zu einer Task, wenn ihm Systemressourcen (CPU, Speicher, Geräte) zugeteilt werden. Im Dialogbetrieb wird ein Auftrag zu einer Task, sobald das LOGON-Kommando akzeptiert ist.

Als Prozesse werden die innerhalb einer Task ablaufenden Aktivitäten, z.B. Programmabläufe, bezeichnet.

Bis heute wird für die Begriffe „Task“ bzw. „Auftrag“ oft synonym der Begriff „Prozeß“ verwendet. In Zukunft sollen die Begriffe so verwendet werden wie oben erklärt. Die Formulierung „bei Prozeßende“ bedeutet also: bei Beendigung eines Programmablaufs.
Mit „Taskende“ ist der Zeitpunkt nach dem LOGOFF-Kommando gemeint. Statt des Begriffs „Prozeßschalter“ wird heute der Begriff „Auftragsschalter“ verwendet.

2 Überblick

Vom Quellprogramm zum ablauffähigen Programm

Damit aus einem COBOL-Quellprogramm ein ablauffähiges Programm wird, sind drei Schritte nötig:

1. Bereitstellen des Quellprogramms (siehe Abschnitt 2.1)
2. Übersetzen: Das Quellprogramm muß in Maschinensprache umgesetzt werden. Der Compiler erzeugt dabei wahlweise ein Objektmodul oder ein Bindelademodul und protokolliert Ablauf und Ergebnis der Übersetzung.
3. Binden: Ein oder mehrere Module werden mit sog. Laufzeitmodulen verknüpft. Es entsteht ein ablauffähiges Programm (siehe Kap. 6).

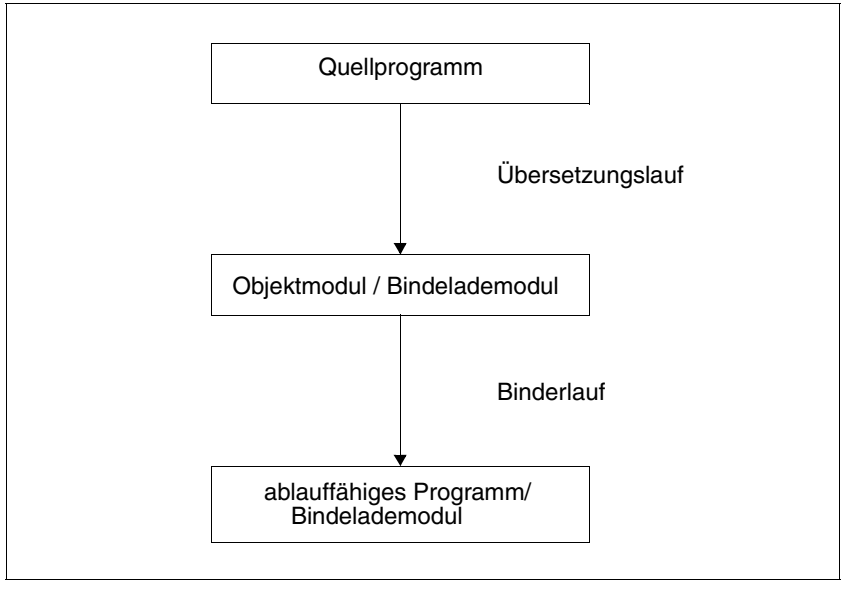


Abbildung 2-1 Der Weg zum ablauffähigen Programm

Der Compiler übernimmt während des Übersetzungslaufs drei Funktionen:

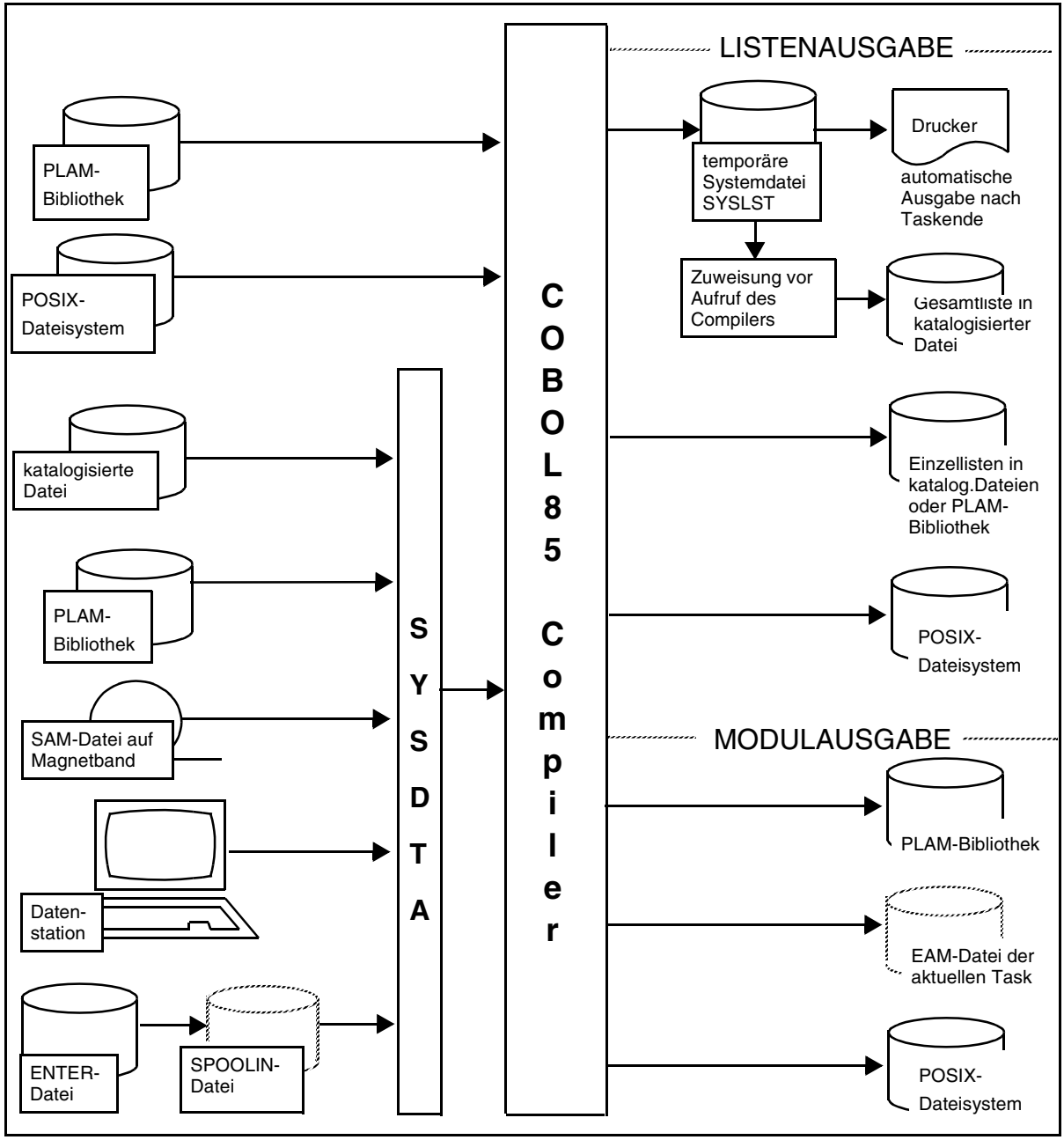
– Überprüfung des Quellprogramms auf syntaktische und semantische Fehler,
– Umsetzung des COBOL-Codes in Maschinensprache,
– Ausgabe von Meldungen, Protokoll-Listen und Modulen.

Durch Steueranweisungen kann der Benutzer

– Funktionen des COBOL85 auswählen,
– die Betriebsmittel für Ein- und Ausgabe zuweisen,
– Eigenschaften der Module bestimmen,
– Art und Umfang der Listenausgabe festlegen.

Die Steuerungsmöglichkeiten, die COBOL85 bzw. das Betriebssystem bieten, werden in den Kapiteln 3 und 4 ausführlich beschrieben.

Mögliche Eingabequellen und Ausgabeziele des Compilers



2.1 Bereitstellen des Quellprogramms

Ein COBOL-Quellprogramm muß nach seiner Codierung dem Compiler für die Übersetzung zugänglich gemacht werden. Unter den verschiedenen Wegen, die dafür zur Verfügung stehen, sind die gebräuchlichsten

– die Eingabe aus einer Datei,

– die Eingabe aus einer PLAM-Bibliothek.

Das Betriebssystem unterstützt die Bereitstellung von Quellprogrammen in Dateien oder PLAM-Bibliotheken durch verschiedene Kommandos und Dienstprogramme.

2.1.1 Bereitstellen in katalogisierten Dateien

COBOL85 kann Quellprogramme aus SAM- oder ISAM-Dateien verarbeiten, wobei ISAM-Dateien mit KEYPOS=5 und KEYLEN=8 katalogisiert sein müssen. Wie das Quellprogramm in eine solche Datei eingegeben werden kann, hängt davon ab, in welcher Form es zur Verfügung steht:

- Liegt das Quellprogramm bereits auf einem externen Datenträger (z.B. Magnetband) gespeichert vor, kann es mit Hilfe geeigneter

  – BS2000-Kommandos (siehe [3]), z.B. des COPY-FILE-Kommandos (für Quellprogramme auf Magnetbändern),

  – Dienstprogramme, z.B. ARCHIVE für Magnetbänder

  in eine katalogisierte Datei übernommen werden.

- Soll das Quellprogramm neu erfaßt werden, läßt sich der Dateiaufbereiter EDT (siehe [21]) einsetzen. Er bearbeitet SAM- oder ISAM-Dateien und stellt Funktionen zur Verfügung, die ein formatgerechtes Erstellen und späteres Ändern von COBOL-Quellprogrammen unterstützen. Dazu gehören u.a.

  – die Möglichkeit, Tabulatoren zu setzen: Sie erlauben ein schnelles und zuverlässiges Positionieren auf die Anfangsspalten der Bereiche A (8.Spalte) und B (12.Spalte) und erleichtern so die Einhaltung des Referenzformats für COBOL-Programme (siehe [1]).

  – Funktionen für das Einfügen, Löschen, Kopieren, Übertragen und Ändern von Quellprogrammzeilen und Zeilen- bzw. Spaltenbereichen,

  – Anweisungen für das Einfügen, Löschen und Ersetzen von Zeichenfolgen in der Datei.

2.1.2 Bereitstellen in PLAM-Bibliotheken

Neben SAM- oder ISAM-Dateien stellen PLAM-Bibliotheken eine weitere wichtige Eingabequelle für den COBOL85-Compiler dar.

Eigenschaften von PLAM-Bibliotheken

PLAM-Bibliotheken sind PAM-Dateien, die mit der Zugriffsmethode PLAM (**P**rimary **L**ibrary **A**ccess **M**ethod) bearbeitet werden (siehe [24]). Für das Einrichten und Verwalten dieser Bibliotheken steht das Dienstprogramm LMS (siehe [10]) zur Verfügung.

Eine PLAM-Bibliothek kann als Elemente nicht nur Quellprogramme oder Quellprogrammteile (COPY-Elemente) sondern z.B. auch Module und ablauffähige Programme enthalten. Die einzelnen Elementarten werden dabei durch Typbezeichnungen charakterisiert.
In einer PLAM-Bibliothek können u.a. Elemente folgender Typen abgelegt werden:

Typbezeichnung	Inhalt der Elemente
S	Quellprogramme, COPY-Elemente
R	Objektmodule oder Großmodule
C	ablauffähige Programme
J	Prozeduren
L	Bindelademodule (LLMs)
P	druckaufbereitete Daten (Listen)

Tabelle 2-1 PLAM-Elementtypen

Eine PLAM-Bibliothek kann auch gleichnamige Elemente enthalten, die sich durch Typ- oder Versionsbezeichnung unterscheiden.

Die Vorteile der Datenhaltung in PLAM-Bibliotheken sind:

– Bis zu 30 % Speicherplatz können durch das Zusammenlegen verschiedener Elementtypen und zusätzliche Komprimierungstechniken eingespart werden.

– Die Zugriffzeiten zu den verschiedenen Elementtypen derselben PLAM-Bibliothek sind kürzer als die Zugriffszeiten bei der herkömmlichen Datenhaltung.

– Der EAM-Speicher wird entlastet, wenn Bindemodule direkt als PLAM-Bibliothekselemente abgelegt werden.

Eingabe in PLAM-Bibliotheken

PLAM-Bibliotheken können Quellprogramme aufnehmen
- aus Dateien,
- aus anderen Bibliotheken,
- über SYSDTA bzw. SYSIPT; d.h. von einer Datenstation oder einer temporären SPOOLIN-Datei.

Wie ein Quellprogramm in eine PLAM-Bibliothek eingegeben werden kann, hängt davon ab, in welcher Form es dafür zur Verfügung steht:

- Liegt es in einer katalogisierten Datei oder als Element einer Bibliothek vor, kann es über das Dienstprogramm LMS in eine PLAM-Bibliothek aufgenommen werden (siehe Beispiel 2-1). Bei der Übernahme eines Quellprogramms aus einer ISAM-Datei mit LMS ist zu beachten, daß mit PAR KEY=YES bzw. SOURCE-ATTRIBUTES=KEEP der ISAM-Schlüssel nicht mitübernommen wird. Ein Quellprogramm mit ISAM-Schlüssel kann der COBOL85-Compiler nicht verarbeiten.
- Soll das Quellprogramm neu erfaßt werden, kann es auch unmittelbar durch den Dateiaufbereiter EDT als Element in eine PLAM-Bibliothek geschrieben werden.

Beispiel 2-1: Übernahme eines Quellprogramms aus einer katalogisierten Datei in eine PLAM-Bibliothek

```
/START-LMS------------------------------------------------------------------(1)
%  LMS0310 LMS VERSION  V03.4B10 STARTED
                                                          PRT=(OUT)
//OPEN-LIBRARY LIB=PLAM.LIB,MODE=UPDATE(STATE=NEW)-----------------------(2)
//ADD-ELEM FROM-FILE=SOURCE.EINXEINS,TO-E=LIB-ELEM(ELEM=EINXEINS,TYPE=S) (3)
//END-------------------------------------------------------------------(4)
%  LMS0311 LMS V03.4B10   TERMINATED NORMALLY
```

(1) Das Dienstprogramm LMS wird aufgerufen.

(2) PLAM.LIB wird als neu einzurichtende (STATE=NEW) Ausgabebibliothek (USAGE=OUT) vereinbart. Sie wird von LMS standardmäßig als PLAM-Bibliothek eingerichtet.

(3) Das Quellprogramm wird aus der katalogisierten Datei SOURCE.EINXEINS als Element vom Typ S unter dem Namen EINXEINS in die PLAM-Bibliothek aufgenommen.

(4) Der LMS-Lauf wird beendet, alle geöffneten Dateien werden geschlossen.

2.2 Quelldaten-Eingabe

Eingaben in den Compiler können folgende Quelldaten sein:
- Quellprogramme (einzelne Quellprogramme oder Quellprogramm-Folgen)
- Quellprogrammteile (COPY-Elemente)
- Compiler-Steueranweisungen (COMOPT-Anweisungen oder SDF-Optionen)

Der Compiler kann Quellprogramme aus katalogisierten SAM- oder ISAM-Dateien, aus Elementen von PLAM-Bibliotheken und aus POSIX-Dateien verarbeiten. Die Bereitstellung von Quellprogrammen ist in Abschnitt 2.1 und 13.2 beschrieben. Die Steueranweisungen für die Eingabe sind in den Kapiteln 3 (Steueranweisungen im SDF-Format) und 4 (COMOPT-Anweisungen) eingehend beschrieben. Die für beide Steuerungsarten gleiche Zuweisung der Systemdatei SYSDTA ist nachfolgend dargestellt.

2.2.1 Zuweisen des Quellprogramms mit dem ASSIGN-SYSDTA-Kommando

Standardmäßig erwartet der Compiler die Quelldaten von der Systemdatei SYSDTA. SYSDTA kann vor dem Aufruf des Compilers einer katalogisierten Datei oder einem Bibliothekselement zugewiesen werden. Das Kommando hierfür lautet:

```
/ASSIGN-SYSDTA [TO-FILE =] { dateiname                                  }
                           { *LIB-ELEM(LIB=bibliothek,ELEM=element)     }
```

Ausführliche Informationen zum ASSIGN-SYSDTA-Kommando können im Handbuch „Kommandos“ [3] nachgelesen werden.

Beispiel 2-2: Einlesen des Quellprogramms aus einer katalogisierten Datei

```
/ASSIGN-SYSDTA QUELL.EINXEINS  ------------------------------------------ (1)
Compileraufruf  ---------------------------------------------------------- (2)
Übersetzung
/ASSIGN-SYSDTA *PRIMARY  ------------------------------------------------- (3)
```

(1) Der Systemdatei SYSDTA wird die katalogisierte Datei QUELL.EINXEINS zugewiesen, in der sich das zu übersetzende Quellprogramm befindet.

(2) Der Compiler wird geladen und gestartet. Er verarbeitet die Daten, die von SYSDTA kommen. Dies gilt nur, falls der Compiler nicht über SDF-Schnittstelle aufgerufen wurde bzw. hier nicht source = ... spezifizert wurde.

(3) Die Systemdatei SYSDTA wird wieder auf ihre Primärzuweisung zurückgesetzt.

Beispiel 2-3: Einlesen eines Quellprogramms aus einer Bibliothek

```
/ASSIGN-SYSDTA *LIBRARY-ELEMENT(LIB=PLAM.LIB,ELEM=BEISP3) ——————— (1)

Compileraufruf  ——————————————————————————————————————————— (2)
Übersetzung

/ASSIGN-SYSDTA *PRIMARY  ——————————————————————————————————— (3)
```

(1) Die Systemdatei SYSDTA wird dem Element BEISP3 in der PLAM-Bibliothek PLAM.LIB zugewiesen.

(2) Der Compiler wird aufgerufen. Er greift über SYSDTA auf das zugewiesene Bibliothekselement zu. Siehe Beispiel voher.

(3) SYSDTA erhält wieder die Primärzuweisung.

Weitere Möglichkeiten der Quelldaten-Eingabe sind an die Steuerung des Compiler mit COMOPT-Anweisungen gebunden und sind in Kapitel 4 beschrieben.

2.2.2 Eingabe von Quellprogrammteilen

Quellprogrammteile (COPY-Elemente) können getrennt von den Quellprogrammen, in denen sie Verwendung finden, in Bibliotheken gespeichert werden. Dies empfiehlt sich vor allem, wenn in verschiedenen Quellprogrammen identische Programmteile vorkommen.
Im Quellprogramm steht stellvertretend für diese Programmteile eine COPY-Anweisung. COPY-Anweisungen dürfen an beliebiger Stelle im Quellprogramm (außer Kommentarzeilen und nicht-numerischen Literalen) stehen.
Stößt der Compiler beim Übersetzen des Quellprogramms auf eine COPY-Anweisung, fügt er aus einer Bibliothek das Element ein, dessen Name in der COPY-Anweisung angegeben wird. Das COPY-Element wird dann so übersetzt, als wäre es im Quellprogramm selbst geschrieben worden. Das Format der COPY-Anweisung ist in Kapitel 11 der COBOL85-Sprachbeschreibung [1] erläutert.

Eingabe von COPY-Elementen aus PLAM-Bibliotheken

Vor dem Aufruf des Compilers müssen die Bibliotheken, in denen sich die COPY-Elemente befinden, dem Compiler mit dem SET-FILE-LINK-Kommando zugewiesen und mit dem im folgenden spezifizierten Linknamen verknüpft werden.
Falls in der COPY-Anweisung ein Bibliotheksname angegeben ist, wird der Linkname aus den ersten 8 Zeichen des Bibliotheksnamens gebildet.
Falls in der COPY-Anweisung kein Bibliotheksname vereinbart wurde, können bis zu zehn Bibliotheken mit den Standard-Linknamen COBLIB, COBLIB1 bis COBLIB9 verknüpft werden. Der Compiler durchsucht dann der Reihe nach die zugewiesenen Bibliotheken, bis er das jeweils gesuchte COPY-Element findet.

Je nach Formulierung der COPY-Anweisung im Quellprogramm sind folgende Verknüpfungen nötig:

COPY-Anweisung		**SET-FILE-LINK-Kommando**	
`COPY textname`		`SET-FILE-LINK [LINK-NAME=]standard-linkname, [FILE-NAME=]libname`	
textname	bis zu 30 Zeichen langer Elementname	standard-linkname	COBLIB COBLIB1..COBLIB9
		libname	Name der katalogisierten Bibliothek, in der das COPY-Element gespeichert ist
`COPY textname OF bibliothek`		`SET-FILE-LINK [LINK-NAME=] linkname, [FILE-NAME=] libname`	
bibliothek	bis zu 30 Zeichen langer Bibliotheksname	linkname	die ersten acht Zeichen von bibliothek
		libname	Name der katalogisierten Bibliothek, in der das COPY-Element gespeichert ist

Eingabe von COPY-Elementen aus dem POSIX-Dateisystem

Wenn das POSIX-Subsystem vorhanden ist, können dem Compiler auch COPY-Texte aus dem POSIX-Dateisystem eingegeben werden. Dies erfolgt mittels einer SDF-P-Variablen mit dem Standardnamen SYSIOL-COBLIB bzw. SYSIOL-bibliotheksname. Je nach Formulierung der COPY-Anweisung im Quellprogramm ist die SDF-P-Variable folgendermaßen zu gestalten (siehe auch Beispiel 2-6, S.18); gilt nicht für BC (Grundausbau):

COPY-Anweisung		**SDF-P-Variable**	
`COPY textname`		`DECL-VAR SYSIOL-COBLIB,INIT=‘*POSIX(pfad)‘, SCOPE=*TASK`	
textname	bis zu 30 Zeichen langer Name der POSIX-Datei, die den COPY-Text enthält. textname darf keine Kleinbuchstaben enthalten.	pfad	Absoluter Pfadname (beginnend mit /) des Dateiverzeichnisses, in dem die Datei textname gesucht werden soll
`COPY textname OF bibliothek`		`DECL-VAR SYSIOL-bibliothek=‘*POSIX(pfad)‘, SCOPE=*TASK`	
bibliothek	bis zu 30 Zeichen langer Name zur Bildung der SDF-P-Variablen mit dem Namen SYSIOL-bibliothek. bibliothek darf keine Kleinbuchstaben enthalten.	pfad	Absoluter Pfadname (beginnend mit /) des Dateiverzeichnisses, in dem die Datei textname gesucht werden soll

Beispiel 2-4: Eingabe zweier COPY-Elemente

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG.
...
    COPY XYZ.------------------------------------------------------------ (1)
    COPY ABC OF BIBLIO. ------------------------------------------------- (2)
...
```

Zuweisung und Verknüpfung:

```
/ASSIGN-SYSDTA BEISPIEL.1 ------------------------------------------------ (3)
/SET-FILE-LINK COBLIB,BIB1 ----------------------------------------------- (4)
/SET-FILE-LINK BIBLIO,BIB2 ----------------------------------------------- (5)
```

Compileraufruf

Das Quellprogramm in der Datei BEISPIEL.1 enthält folgende COPY-Anweisungen:

(1) XYZ ist der Name des Elements, unter dem das COPY-Element in der PLAM-Bibliothek BIB1 abgespeichert ist.

(2) ABC ist der Name des Elements, unter dem das COPY-Element in der PLAM-Bibliothek BIB2 mit dem Linknamen BIBLIO abgespeichert ist.

(3) SYSDTA wird der Datei BEISPIEL.1 zugewiesen. Von dort erhält der Compiler das Quellprogramm, in dem zwei COPY-Anweisungen stehen.

(4) Das erste SET-FILE-LINK-Kommando weist die PLAM-Bibliothek BIB1 zu und verknüpft sie mit dem Standard-Linknamen COBLIB.

(5) Das zweite SET-FILE-LINK-Kommando weist die PLAM-Bibliothek BIB2 zu und verknüpft sie mit dem in der COPY-Anweisung angegebenen Linknamen BIBLIO.

Beispiel 2-5: Eingabe mehrerer COPY-Elemente aus verschiedenen Bibliotheken

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1.
...
    COPY A1.    )
    COPY B1.    }  ------------------------------------------------------(1)
    COPY D1.    )
...
Zuweisung und Verknüpfung:
/ASSIGN-SYSDTA BEISPIEL2  ------------------------------------------------(2)

/SET-FILE-LINK COBLIB,A     )
/SET-FILE-LINK COBLIB1,B    }  ------------------------------------------(3)
/SET-FILE-LINK COBLIB3,D    )

Compileraufruf  ----------------------------------------------------------(4)
```

Das Quellprogramm BEISPIEL2 enthält folgende COPY-Anweisungen:

(1) A1, B1, D1 sind die Namen der COPY-Elemente, unter denen sie in den katalogisierten Bibliotheken A, B, D gespeichert sind.

(2) SYSDTA wird der katalogisierten Datei BEISPIEL2 zugewiesen. Von dort erhält der Compiler das Quellprogramm, in dem drei COPY-Anweisungen stehen.

(3) Die Bibliotheken A, B und D werden zugewiesen und mit den Standard-Linknamen verknüpft. Dabei muß der Standard-Linkname COBLIB stets zugewiesen werden, während die Verknüpfung mit COBLIB1 bis COBLIB9 in Anzahl und Reihenfolge beliebig ist.

(4) Nach dem Aufruf durchsucht der Compiler COBLIB, COBLIB1 und COBLIB3 in dieser Reihenfolge nach den in den COPY-Anweisungen genannten Elementen und kopiert die Programmzeilen in die Quellprogrammdatei BEISPIEL2.

Beispiel 2-6: Eingabe eines COPY-Elements aus dem POSIX-Dateisystem

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1.
...
COPY ATEXT. ------------------------------------------------------------------ (1)
...
```

Zuweisung des POSIX-Dateisystems durch Einrichten und Setzen einer SDF-P-Variablen:

```
/DECL-VAR SYSIOL-COBLIB ,INIT='*POSIX(/usr/dir1),*POSIX(/usr/dir2)',
                              SCOPE=*TASK ------------------------------- (2)

/START-COBOL85-COMPILER? ------------------------------------------------- (3)
```

(1) Das COPY-Element ATEXT befindet sich als Datei im POSIX-Dateisystem.

(2) Mit dem SDF-P-Kommando DECL-VARIABLE wird die Variable auf die Pfade im POSIX gesetzt, in deren Verzeichnissen dir1 und dir2 nach der Datei ATEXT gesucht werden soll.

(3) Der Zugriff auf das POSIX-Dateisystem ist nur möglich, wenn der Compiler mit SDF-Steuerung aufgerufen wird. Mit dem an das Aufrufkommando angehängten „?“ gelangt der Benutzer in den SDF-Menümodus (siehe Abschnitt 3.1.2, S.27), in dem weitere Angaben zur Steuerung des Übersetzungslaufs erfolgen können.

(4) Der Compiler akzeptiert COPY-Elemente aus dem POSIX-Dateisystem nur, wenn ihre Dateinamen ausschließlich aus Großbuchstaben bestehen.

2.3 Ausgaben des Compilers

2.3.1 Ausgabe von Modulen

Der Compiler übersetzt die eingegebenen Quelldaten in Maschinensprache und erzeugt auf diese Weise ein oder mehrere Objektmodule (OM Format) oder Bindelademodule (LLM Format). Der Benutzer kann veranlassen, daß einem Modul ein Symbolisches Adreßbuch (LSD, **L**ist for **S**ymbolic **D**ebugging) zugeordnet wird, das die symbolischen Adressen des Quellprogramms speichert.

Objektmodule gibt der Compiler standardmäßig in die temporäre EAM-Datei der aktuellen Task aus. Die Objektmodule werden dort additiv, d.h. ohne Bezug zueinander, abgespeichert.
Die EAM-Datei gehört zu der Task, in der die Übersetzung stattfindet. Sie wird beim ersten Übersetzungslauf für diese Task angelegt und bei Task-Ende (LOGOFF-Bearbeitung) automatisch gelöscht. Soll das Ergebnis der Übersetzung also weiterverwendet werden, so ist der Benutzer dafür verantwortlich, daß der Inhalt der EAM-Datei sichergestellt bzw. weiterverarbeitet wird. Für die Sicherstellung von Objektmoduln aus der EAM-Datei in PLAM-Bibliotheken steht ihm dabei das Dienstprogramm LMS zur Verfügung (siehe [10]).

Werden die übersetzten Objektmodule in der EAM-Datei nicht mehr benötigt, z.B. weil das Quellprogramm noch zu korrigierende Fehler enthält, so empfiehlt es sich, die EAM-Datei spätestens vor dem nächsten Übersetzungslauf mit dem Kommando

```
DELETE-SYSTEM-FILE [FILE-NAME=] OMF
```

zu löschen.

Bindelademodule (LLMs) schreibt der Compiler grundsätzlich als Elemente vom Typ L in eine PLAM-Bibliothek.

Falls das POSIX-Subsystem vorhanden ist, können die Module ins POSIX-Dateisystem ausgegeben werden. Diese Möglichkeit ist in Abschnitt 3.3.5 (MODULE-OUTPUT-Option, S.42) beschrieben.

Bildung von Elementnamen bei der Ausgabe von Modulen in Bibliotheken

	Modulformat OM	**Modulformat LLM**
Quellprogramm	Standardname abgeleitet aus	
nicht gemeinsam benutzbarer Code[1)]		
nicht segmentiert	PROGRAM-ID-Name 1..8[2)]	PROGRAM-ID-Name 1..30[3)]
segmentiert	PROGRAM-ID-Name 1..6 + Segmentnummer (für jedes Segment)	PROGRAM-ID-Name 1..30[3)] (Segmentierung ignoriert)
gemeinsam benutzbarer Code[4)]	PROGRAM-ID-Name 1..7@ (Code-Modul) PROGRAM-ID-Name 1..7 (Datenmodul)	PROGRAM-ID-Name 1..30[3)]

Tabelle 2-2 Elementnamenbildung bei Modulausgabe

1) Modul erzeugt mit
COMPILER-ACTION=MODULE-GENERATION(SHAREABLE-CODE=NO)
bzw. COMOPT GENERATE-SHARED-CODE=NO

2) Der Name sollte in den ersten 7 Zeichen eindeutig sein.

3) Statt des Standardnamens kann mit
MODULE-OUTPUT=*LIBRARY-ELEMENT(LIBRARY=<filename>,
ELEMENT=<composed-name>)
bzw. COMOPT MODULE-ELEMENT=elementname
ein eigener Elementname gewählt werden.
Diese Option beeinflußt jedoch nicht den Namen des Einsprungpunktes, d.h. den Namen, der in der CALL-Anweisung angegeben wird.

4) Modul erzeugt mit
COMPILER-ACTION=MODULE-GENERATION(SHAREABLE-CODE=YES)
bzw. COMOPT GENERATE-SHARED-CODE=YES

2.3.2 Ausgabe von Listen und Meldungen

Ausgabe von Listen

Der Compiler kann folgende Protokoll-Listen des Übersetzungslaufs erzeugen:

Steueranweisungsliste	OPTION LISTING
Quellprogrammliste	SOURCE LISTING
Bibliotheksliste	LIBRARY LISTING
Objektliste	OBJECT PROGRAM LISTING
Adreßliste Querverweisliste	LOCATOR MAP LISTING
Fehlermeldungsliste	DIAGNOSTIC LISTING

Standardmäßig schreibt der Compiler jede angeforderte Liste in eine eigene katalogisierte Datei. Die Listen in den katalogisierten Dateien können zu einem beliebigen Zeitpunkt mit Hilfe des PRINT-FILE-Kommandos (siehe [3]) ausgedruckt werden.

Statt in katalogisierte Dateien können die angeforderten Listen auch als Elemente in eine PLAM-Bibliothek geschrieben werden.

Der Benutzer kann mit einer entsprechenden Steueranweisung veranlassen, daß die angeforderten Listen auf die Systemdatei SYSLST ausgegeben werden. Die dabei erzeugte temporäre Datei gibt das System automatisch auf den Drucker aus.

Die Erzeugung und Ausgabe der Protokoll-Listen kann der Benutzer steuern mit

– der SDF-Option LISTING (siehe Kapitel 3) oder

– den COMOPT-Anweisungen LISTFILES, LIBFILES oder SYSLIST (siehe Kapitel 4).

Falls das POSIX-Subsystem vorhanden ist, können die Listen (außer der Objektliste) ins POSIX-Dateisystem ausgegeben werden. Diese Möglichkeit ist in Abschnitt 3.3.6, LISTING-Option, S.45, beschrieben.

Ausgabe von Meldungen

Die Meldungen des Compilers über den Ablauf der Übersetzung (COB90xx) werden standardmäßig über die Systemdatei SYSOUT auf die Datensichtstation ausgegeben.
Kapitel 14 enthält die kommentierten Texte aller vom Compiler ausgegebenen COB90xx-Meldungen.

2.4 Steuerungsmöglichkeiten des Compilers

Die Quelldaten-Eingabe, die Eigenschaften des Moduls, die Ausgabe von Meldungen und Listen sowie die Ausgabe des Moduls lassen sich durch Anweisungen an den COBOL85 steuern.
Der COBOL85-Compiler kann auf zwei Arten gesteuert werden:

- durch Optionen im SDF-Syntaxformat

  oder

- durch COMOPT-Anweisungen.

Der Benutzer entscheidet sich mit der Gestaltung des Compiler-Aufrufkommandos für eine der beiden Steuerungsarten:

Aufrufkommandos	**Steuerungsart**
/START-COBOL85-COMPILER optionen	SDF-Steuerung, Expert-Modus
/?	SDF-Steuerung, Menü-Modus
/START-COBOL85-COMPILER?	SDF-Steuerung, Menü-Modus
/START-PROGRAM *name Compilerphase*	COMOPT-Steuerung
/START-COBOL85-COMPILER	keine, Eingabe des Quellprogramms von SYSDTA

Tabelle 2-3 Aufrufkommando und Steuerungsart

Die SDF-Steuerung ist in Kapitel 3, die COMOPT-Steuerung in Kapitel 4 ausführlich beschrieben.

Die Steuerung des Compilers im POSIX-Subsystem ist in Kapitel 13 beschrieben.

2.5 Beendigung des Compilerlaufs

Das Beendigungsverhalten des COBOL85-Compilers hängt davon ab,

– welcher Klasse die im Quellprogramm erkannten Fehler angehören,

– ob der Compiler selbst fehlerfrei abläuft.

Dieses Verhalten ist vor allem dann von Bedeutung, wenn der COBOL85-Compiler in einer Prozedur aufgerufen oder von Monitor-Jobvariablen überwacht wird.
Die folgende Tabelle gibt einen Überblick über die möglichen Fälle, deren Auswirkung auf den weiteren Ablauf der Prozedur und den Inhalt der Rückkehrcode-Anzeige der Monitor-Jobvariablen:

Fehler	Beendigung	Dump	Rückkehrcode-Anzeige in Monitor-Jobvariablen	Verhalten in Prozeduren
keine Fehler	normal	nein	0000	keine Verzweigung
Fehlerklasse F	normal	nein	0001	
Fehlerklasse I	normal	nein	0001	
Fehlerklasse 0	normal	nein	1002	
Fehlerklasse 1	normal	nein	1003	
Fehlerklasse 2	normal	nein	2004	keine Verzweigung außer bei expliziter Unterdrückung der Listen- bzw. Modul-erzeugung
Fehlerklasse 3	normal	nein	2005	Verzweigung zum nächsten STEP-, ABEND-, ABORT-, ENDP- oder LOGOFF-Kommando
Compilerfehler	abnormal	ja	3006	

Tabelle 2-4 Beendigungsverhalten des Compilers

2.6 Übersetzung von Quellprogramm-Folgen

Für die Übersetzung von Quellprogramm-Folgen (sequence of programs) gelten einige Besonderheiten:

Steueranweisungen:
Die vor dem Aufruf des Compilers angegebenen Steueranweisungen gelten für alle Programme der Quellprogramm-Folge.
Zwischen den Programmen einer Folge dürfen keine Steueranweisungen stehen.

Listenausgabe über SYSLST:
Die angeforderten Listen werden in eine einzige SPOOL-Datei ausgegeben, in der sie programmspezifisch nacheinander aufgeführt sind.

Listenausgabe in katalogisierte Dateien:
Bei Verwendung der Standardnamen werden für jedes Programm der Quellprogramm-Folge ebensoviele Dateien angelegt, wie Listen angefordert wurden.
Bei Verwendung der Standard-Linknamen werden Dateien nach Listenarten angelegt.
Die mit OPTLINK verknüpfte Datei enthält die Optionenlisten aller Programme der Folge, die mit SRCLINK verknüpfte Datei alle Quellprogrammlisten, die mit ERRLINK verknüpfte Datei alle Fehlerlisten, die mit LOCLINK verknüpfte Datei alle Adreß-/Querverweislisten.

Listenausgabe in eine PLAM-Bibliothek:
Für jedes Programm der Quellprogrammfolge werden ebensoviele Elemente angelegt, wie Listen angefordert wurden.

Versorgen der Monitor-Jobvariablen:
In der Monitor-Jobvariablen wird stets der Rückkehrcode für dasjenige Quellprogramm angezeigt, das den Fehler mit dem höchsten Gewicht enthält.

Compilerabbruch:
Tritt in einem Programm innerhalb einer Quellprogramm-Folge ein Fehler auf, der zum Abbruch der Übersetzung dieses Programms führt, so wird der gesamte Compilerlauf beendet; d.h. alle nachfolgenden Programme werden nicht mehr übersetzt.

Modulausgabe:
Für jedes Quellprogramm einer Folge wird ein Modul erzeugt. In die EAM-Datei werden die Module nacheinander abgelegt, in eine PLAM-Bibliothek als einzelne Elemente.

3 Steuerung des Compilers über SDF

Der COBOL85-Compiler kann über SDF (**S**ystem **D**ialog **F**acilities) gesteuert werden.

In den folgenden Abschnitten werden die wesentlichen Vorgehensweisen im Umgang mit SDF beschrieben. Die ausführliche Darstellung der Dialog-Schnittstelle SDF findet sich in den Handbüchern „Dialogschnittstelle SDF“ [5] und „Kommandos“ [3].

3.1 Compileraufruf und Eingabe der Optionen

Im Dialogbetrieb bietet SDF folgende Möglichkeiten:

- Eingabe von der Datensichtstation ohne Benutzerführung, nachfolgend „Expert-Modus“ genannt.
- Eingabe von der Datensichtstation mit Benutzerführung in drei verschiedenen Stufen, nachfolgend „Menü-Modus“ genannt.

3.1.1 SDF-Expert-Modus

Nach dem LOGON-Kommando ist standardmäßig der SDF-Expert-Modus eingeschaltet. In diesem Modus startet der Benutzer den Übersetzungslauf folgendermaßen:

```
/START-COBOL85-COMPILER optionen

Abkürzung: S-COB-C optionen
```

Die Übersetzung wird sofort nach Eingabe des Kommandos gestartet.

Falls keine Optionen angegeben werden, liest der Compiler das Quellprogramm von SYSDTA, sofern SYSDTA der Datei bzw. dem Bibliothekselement zugewiesen ist, die das Quellprogramm enthält (siehe 2.2.1).

Für die Optioneneingabe im Expert-Modus gilt allgemein:

- Alle Optionen, Parameter und Operandenwerte müssen durch Kommas voneinander getrennt werden.
- Reicht für die Optioneneingabe eine Zeile nicht aus, stehen zwei Möglichkeiten zur Verfügung:
  - Mit einem Bindestrich („-“) nach dem zuletzt eingegebenen Zeichen können Fortsetzungszeilen erzeugt werden.
  - Alle Optionen können fortlaufend (d.h. ohne Rücksicht auf das Zeilenende) geschrieben werden.

Optionen können als Schlüsselwort- oder als Stellungsoperanden angegeben werden:

- Schlüsselwort-Operanden

  Die Schlüsselwörter müssen formatgetreu angegeben werden, können aber soweit abgekürzt werden, daß sie innerhalb der jeweiligen SDF-Umgebung eindeutig sind. Unzulässige Abkürzungen und Schreibfehler werden als Syntaxfehler gemeldet und können sofort korrigiert werden.

Beispiel 3-1

```
/START-COBOL85-COMPILER SOURCE=BSP,-
/COMPILER-ACTION=MODULE-GENERATION(SHAREABLE-CODE=YES),-
/ACTIVATE-FLAGGING=ANS85,-
/TEST-SUPPORT=AID
```

Die maximal mögliche Abkürzung in der Beispielumgebung lautet:

```
/S-COB-C S=BSP,C-A=MOD(SH=Y),A-F=ANS85,T=A
```

- Stellungsoperanden

  Die Operanden-Schlüsselwörter (d.h. jene Schlüsselwörter, die im Format links vom Gleichheitszeichen stehen) und das Gleichheitszeichen können weggelassen werden, sofern die festgelegte Reihenfolge der Operanden und ihrer Werte exakt eingehalten wird. Alle Operanden, die nicht angegeben werden, weil ihre Voreinstellung gelten soll, müssen durch das Trennzeichen "," (Komma) markiert werden.
  Folgen auf die zuletzt belegte Option noch weitere mögliche Optionen, braucht deren Position nicht durch Trennzeichen angegeben zu werden.

  In Prozeduren sollten die Optionen nicht als Stellungsoperanden angegeben werden.

3.1.2 SDF-Menü-Modus

Es gibt zwei Möglichkeiten, den SDF-Menü-Modus zu verwenden:

Permanenter Menü-Modus

Mit dem SDF-Kommando
/MODIFY-SDF-OPTIONS GUIDANCE = MAXIMUM / MEDIUM / MINIMUM
gelangt der Benutzer in das SDF-Hauptmenü. Die verfügbaren Compiler-Aufrufkommandos findet er dort unter dem Stichwort PROGRAMMING-SUPPORT. Mit der Angabe der zugehörigen Nummer in der Eingabezeile wird das PROGRAMMING-SUPPORT-Menü ausgegeben. Von dort aus kann dann der Compiler unter Angabe der Kommando-Nummer aufgerufen werden.
Die Werte des MODIFY-SDF-OPTIONS-Kommandos bedeuten:

MAXIMUM	Maximale Hilfestufe, d.h. sämtliche Operandenwerte mit Zusätzen, Hilfetexte für Kommandos und Operanden.
MEDIUM	Sämtliche Operandenwerte ohne Zusätze, Hilfetexte nur für Kommandos.
MINIMUM	Minimale Hilfestufe, d.h. nur Standardwerte der Operanden, keine Zusätze, keine Hilfetexte.

Im permanenten Menü-Modus befindet sich der Benutzer solange, bis er mit dem Kommando MODIFY-SDF-OPTION GUIDANCE=EXPERT explizit in den Expert-Modus zurückschaltet.

Temporärer Menü-Modus

Für die Compilersteuerung im temporären Menü-Modus gibt es zwei Wege:

1. Schrittweises Durchlaufen der SDF-Menüs bis zum Operandenfragebogen

   Mit der Angabe des Fragezeichens auf Systemebene gelangt der Benutzer in das SDF-Hauptmenü.

```
/?
       Wechsel in das SDF-Hauptmenü
    Angabe der Nummer des PROGRAMMING-SUPPORT-Menüs
       Wechsel in das PROGRAMMING-SUPPORT-Menü
    Angabe der Nummer des Compiler-Aufrufkommandos
       Wechsel in den Operandenfragebogen
```

2. Unmittelbarer Wechsel in den Operandenfragebogen

   Unmittelbar an START-COBOL85-COMPILER wird ein Fragezeichen angehängt:

```
/START-COBOL85-COMPILER? [optionen]

       Wechsel in den Operandenfragebogen
```

   Mit START-COBOL85-COMPILER? verzweigt die Steuerung in den Menü-Modus, und die erste Seite des Operandenfragebogens wird aufgeschlagen.
   Der Fragebogen enthält ggf. die Operandenwerte der Optionen, die unmittelbar nach START-COBOL85-COMPILER? angegeben wurden.

   Durch Angabe von *CANCEL in der NEXT-Zeile bzw. durch Betätigen der K1-Taste kann der Benutzer aus jedem Menü sofort zurück in den Expert-Modus gelangen.

   Nach der Übersetzung befindet sich der Benutzer wieder im Expert-Modus (angezeigt durch "/").

Hinweise zur Bearbeitung des Operandenfragebogens

Der Operandenfragebogen ist weitgehend selbsterklärend aufgebaut. Bei der Bearbeitung ist vor allem zu beachten, daß allein der Eintrag in der Eingabezeile ("NEXT:...") den Ausschlag gibt, welche Operation ausgeführt wird. Die jeweils zulässigen Eingaben sind unter dieser Zeile aufgeführt.

Im folgenden sind die wichtigsten Steuerzeichen zur Bearbeitung des Operandenfragebogens zusammengefaßt.
Die ausführliche Beschreibung des optimalen Umgangs mit SDF findet sich im Handbuch „Dialogschnittstelle SDF“ [5].

Steuerzeichen zur Bearbeitung des Operandenfragebogens

?	als Operandenwert liefert Hilfetext und Angabe des Wertebereichs für diesen Operanden. Hat SDF nach vorheriger fehlerhafter Eingabe die Meldung "CORRECT INCORRECT OPERANDS" gebracht, liefert das Fragezeichen zusätzliche detaillierte Fehlermeldungen. Der Zeilenrest muß nicht gelöscht werden.
!	als Operandenwert setzt für diesen Operanden den Standardwert wieder ein, wenn der abgebildete Standardwert vorher überschrieben wurde. Der Zeilenrest muß nicht gelöscht werden.
<operand>(	Geöffnete Klammer nach einem struktureinleitenden Operanden gibt den Unterfragebogen für die zugehörige Struktur aus. Nach der geöffneten Klammer angegebene Operanden werden im Unterfragebogen abgebildet.
–	als letztes Zeichen in einer Eingabezeile bewirkt die Ausgabe einer Fortsetzungszeile (bis zu 9 Fortsetzungszeilen pro Operand möglich).
Line-(LZF)Taste	löscht ab der Schreibmarke alle Zeichen der Eingabezeile.

3.2 SDF-Syntaxbeschreibung

In den folgenden Tabellen wird die Metasyntax der Optionenformate erläutert.

Tabelle 3-1: Metazeichen

In den Optionenformaten werden bestimmte Zeichen und Darstellungsformen verwendet, deren Bedeutung in der folgenden Tabelle erläutert wird.

Zeichen	Bedeutung	Beispiele
GROSSBUCHSTABEN	Großbuchstaben bezeichnen Schlüsselwörter. Einige Schlüsselwörter beginnen mit *	LISTING = STD SOURCE = *SYSDTA
=	Das Gleichheitszeichen verbindet einen Operandennamen mit dem dazugehörenden Operandenwert.	LINE-SIZE = 132
< >	Spitze Klammern kennzeichnen Variablen, deren Wertevorrat durch Datentypen und ihre Zusätze beschrieben wird (siehe Tabellen 3-2 und 3-3).	... = <integer 1..100>
Unterstreichung	Die Unterstreichung kennzeichnet den Standardwert eines Operanden.	MODULE-LIBRARY = *OMF
/	Der Schrägstrich trennt alternative Operandenwerte.	SHAREABLE-CODE = NO / YES
(...)	Runde Klammern kennzeichnen Operandenwerte, die eine Struktur einleiten.	TEST-SUPPORT = AID(...)
Einrückung |	Die Einrückung kennzeichnet die Abhängigkeit zu dem jeweils übergeordneten Operanden. Der Strich kennzeichnet zusammengehörende Operanden einer Struktur. Sein Verlauf zeigt Anfang und Ende einer Struktur an. Innerhalb einer Struktur können weitere Strukturen auftreten. Die Anzahl senkrechter Striche vor einem Operanden entspricht der Strukturtiefe.	LISTING = PARAMETERS(...) PARAMETERS(...) | SOURCE = YES(...) | YES(...) | | COPY-EXP... | | . | | .
,	Das Komma steht vor weiteren Operanden der gleichen Strukturstufe.	,SHAREABLE-CODE = ,PREPARE-CANCEL-STMT =

Tabelle 3-2: Datentypen

Variable Operandenwerte werden in SDF durch Datentypen dargestellt. Jeder Datentyp repräsentiert einen bestimmten Wertevorrat. Die Anzahl der Datentypen ist beschränkt auf die in Tabelle 3-2 beschriebenen Datentypen.

Die Beschreibung der Datentypen gilt für alle Optionen. Deshalb werden bei den entsprechenden Operandenbeschreibungen nur noch Abweichungen von Tabelle 3-2 erläutert.

Datentyp	Zeichenvorrat	Besonderheiten
alphanum-name	A…Z 0…9 $, #, @	
composed-name	A…Z 0…9 $, #, @ Bindestrich Punkt	alphanumerische Zeichenfolge, die in mehrere durch Punkt oder Bindestrich getrennte Teilzeichenfolgen gegliedert sein kann
c-string	EBCDIC-Zeichen	In Hochkommas eingeschlossene Folge von EBCDIC-Zeichen. Der Buchstabe C kann vorangestellt werden.
filename	A…Z 0…9 $, #, @ Bindestrich Punkt	Eingabeformat: :cat:$user.{ datei | datei(nr) | gruppe | gruppe{ (*abs) | (+rel) | (-rel) } } :cat: wahlfreie Angabe der Katalogkennung; Zeichenvorrat auf A…Z und 0…9 eingeschränkt; max. 4 Zeichen; ist in Doppelpunkte einzuschließen; Standardwert ist die Katalogkennung, die der Benutzerkennung laut Eintrag im Benutzerkatalog zugeordnet ist. $user. wahlfreie Angabe der Benutzerkennung; Zeichenvorrat ist A…Z, 0…9, $, #, @; max. 8 Zeichen; darf nicht mit einer Ziffer beginnen; $ und Punkt müssen angegeben werden; Standardwert ist die eigene Benutzerkennung.

Datentyp	Zeichenvorrat	Besonderheiten
filename (Forts.)		$. (Sonderfall) System-Standardkennung datei Datei- oder Jobvariablenname; letztes Zeichen darf kein Bindestrich oder Punkt sein; max. 41 Zeichen; muß mindestens ein Zeichen aus A…Z enthalten. #datei (Sonderfall) @datei (Sonderfall) # oder @ als erstes Zeichen kennzeichnet je nach Systemgenerierung temporäre Dateien und Jobvariablen. datei(nr) Banddateiname nr: Versionsnummer; Zeichenvorrat ist A…Z, 0…9, $, #, @. Klammern müssen angegeben werden. gruppe Name einer Dateigenerationsgruppe (Zeichenvorrat siehe unter "datei") gruppe{(*abs) (+rel) (-rel)} (*abs) absolute Generationsnummer (1-9999); * und Klammern müssen angegeben werden. (+rel) (-rel) relative Generationsnummer (0-99); Vorzeichen und Klammern müssen angegeben werden.
integer	0…9, +, -	+ bzw. - kann nur erstes Zeichen sein.

Tabelle 3-3: Zusätze zu Datentypen

Zusätze zu Datentypen kennzeichnen weitere Eingabevorschriften für Datentypen.
Die Zusätze schränken den Wertevorrat ein oder erweitern ihn. Im Handbuch werden folgende Zusätze in gekürzter Form dargestellt:

generation	gen
cat-id	cat
user-id	user
version	vers

Die Beschreibung der Zusätze zu den Datentypen gilt für alle Optionen und Operanden. Deshalb werden bei den entsprechenden Operandenbeschreibungen nur noch Abweichungen von Tabelle 3-3 erläutert.

Zusatz	**Bedeutung**
x..y	Längenangabe x Mindestlänge für den Operandenwert; x ist eine ganze Zahl. y Maximallänge für den Operandenwert; y ist eine ganze Zahl. x=y Der Operandenwert muß genau die Länge x haben.
with-low	Kleinbuchstaben zulässig
without	Schränkt die Angabemöglichkeiten für einen Datentyp ein.
-gen	Die Angabe einer Dateigeneration oder Dateigenerationsgruppe ist nicht erlaubt.
-vers	Die Angabe der Version (siehe datei(nr)) ist bei Banddateien nicht erlaubt.
-cat	Die Angabe einer Katalogkennung ist nicht erlaubt.
-user	Die Angabe einer Benutzerkennung ist nicht erlaubt.

3.3 SDF-Optionen zur Steuerung des Übersetzungslaufs

Name der Option	Zweck
SOURCE	Bestimmen der Eingabequelle des Quellprogramms
SOURCE-PROPERTIES	Festlegen bestimmter Eigenschaften des Quellprogramms
ACTIVATE-FLAGGING	Kennzeichnung bestimmter Sprachelemente in der Fehlerliste mit einer Meldung der Klasse F
COMPILER-ACTION	Teilweise Durchführung des Compilerlaufs; Beeinflussen einiger Eigenschaften des generierten Codes sowie des Modulformats (Objektmodul, LLM).
MODULE-OUTPUT	Bestimmen des Namens und Ausgabezieles der Objektmodule oder LLMs
LISTING	Festlegen, welche Listen ausgegeben werden, welches Layout die Listen haben und wohin sie ausgegeben werden sollen
TEST-SUPPORT*	Bestimmen, ob Informationen für die Testhilfe AID erzeugt werden sollen
OPTIMIZATION	Ein-Ausschalten der Optimierung des Compilers
RUNTIME-CHECKS	Aktivieren der Prüfroutinen des Laufzeitsystems
COMPILER-TERMINATION	Bestimmen, ab welcher Fehlerzahl der Übersetzungslauf abgebrochen werden soll
MONJV	Einrichten einer Jobvariablen zur Überwachung des Compilerlaufs
RUNTIME-OPTIONS	Festlegen einiger Ablaufeigenschaften des Programms

Übersicht: Die Optionen zur Steuerung des Compilers

* Option in COBOL85-BC nicht verfügbar

3.3.1 SOURCE-Option

Die Parameter dieser Option bestimmen, ob das Quellprogramm von SYSDTA, aus einer katalogisierten BS2000-Datei, aus einer PLAM-Bibliothek oder aus einer POSIX-Datei eingelesen wird.

Format

```
SOURCE = *SYSDTA / <filename 1..54> / <c-string 1..1024 with-low> / *LIBRARY-ELEMENT(...)
  *LIBRARY-ELEMENT(...)
      LIBRARY = <filename 1..54>
     ,ELEMENT = <composed-name 1..40>(...)
        <composed-name>(...)
            VERSION = *HIGHEST-EXISTING / *UPPER-LIMIT / <alphanum-name 1..24>
```

SOURCE = *SYSDTA
Das Quellprogramm wird von der Systemdatei SYSDTA eingelesen, die im Dialogbetrieb standardmäßig der Datensichtstation zugewiesen ist. Wurde SYSDTA vor Beginn des Übersetzungslaufs mit dem ASSIGN-SYSDTA-Kommando der Quellprogrammdatei zugewiesen, erübrigt sich die Angabe der SOURCE-Option.

SOURCE = <filename 1..54>
Mit <filename> wird eine katalogisierte Datei zugewiesen. Nach der Übersetzung existiert ein TFT-Eintrag für den Linknamen SRCFILE, der mit dem Dateinamen <filename> verknüpft ist.

SOURCE = <c-string 1..1024 with-low>
Wenn das POSIX-Subsystem zugreifbar ist, kann mit diesem Parameter eine Quelldatei aus dem POSIX-Dateisystem angefordert werden. Mit <c-string> wird der Name der POSIX-Datei angegeben. Enthält <c-string> keinen Dateiverzeichnisnamen, sucht der Compiler die Quelldatei unter dem angegebenen Dateinamen im Home-Dateiverzeichnis der aktuellen BS2000-Benutzerkennung. Steht die Datei in einem anderen Dateiverzeichnis, muß mit <c-string> der absolute Pfadname angegeben werden.

Dieser Operand ist in COBOL-BC nicht verfügbar.

SOURCE = *LIBRARY-ELEMENT(...)
Mit diesem Parameter wird eine PLAM-Bibliothek und ein Element daraus angegeben.

LIBRARY = <filename 1..54>
Name der PLAM-Bibliothek, in der das Quellprogramm als Element steht. Nach der Übersetzung existiert ein TFT-Eintrag für den Linknamen SRCLIB, der mit dem Namen <filename> der PLAM-Bibliothek verknüpft ist.

ELEMENT = <composed-name 1..40>(...)
Name des Bibliothekselements, in dem das Quellprogramm steht.

VERSION = *HIGHEST-EXISTING / *UPPER-LIMIT /
<alphanum-name 1..24>
Versionsbezeichnung des Bibliothekselements. Wird keine Version oder *HIGHEST-EXISTING angegeben, liest der Compiler die Version des Elements mit der höchsten in der Bibliothek vorhandenen Versionsbezeichnung.
Wird *UPPER-LIMIT angegeben, liest der Compiler die Version des Elements mit der größtmöglichen Versionsnummer (vom LMS angezeigt mit "@").

3.3.2 SOURCE-PROPERTIES-Option

Mit dieser Option können bestimmte Eigenschaften des Quellprogramms festgelegt werden.

Format

```
SOURCE-PROPERTIES = STD / PARAMETERS(...)
    PARAMETERS(...)
        RETURN-CODE = FROM-COBOL-SUBPROGRAMS / FROM-ALL-SUBPROGRAMS
```

SOURCE-PROPERTIES = STD
Es wird der voreingestellte Wert der nachfolgenden PARAMETERS-Struktur übernommen.

SOURCE-PROPERTIES = PARAMETERS(...)

RETURN-CODE = FROM-COBOL-SUBPROGRAMS
Das Sonderregister RETURN-CODE wird nur zum Informationsaustausch zwischen den COBOL-Programmen einer Ablaufeinheit verwendet.

RETURN-CODE = FROM-ALL-SUBPROGRAMS
Das Sonderregister RETURN-CODE soll auch zur Aufnahme eines Funktionswertes aus einem C-Unterprogramm dienen.

3.3.3 ACTIVATE-FLAGGING-Option

Diese Option veranlaßt den Compiler, bestimmte Sprachelemente gemäß ANS85 oder gemäß "Federal Information Processing Standard" (FIPS) in der Fehlerliste mit einer Meldung der Klasse F zu kennzeichnen.

Format

```
ACTIVATE-FLAGGING = NO / ANS85 / FIPS(...)
    FIPS(...)
        |  OBSOLETE-FEATURES = NO / YES
        | ,NONSTANDARD-LANGUAGE = NO / YES
        | ,ABOVEMIN-SUBSET = NO / YES
        | ,ABOVEINTERMED-SUBSET = NO / YES
        | ,REPORT-WRITER = NO / YES
        | ,ALL-SEGMENTATION = NO / YES
        | ,SEGMENTATION-ABOVE1 = NO / YES
        | ,INTRINSIC-FUNCTIONS = NO / YES
```

ACTIVATE-FLAGGING = NO
Es werden keine Sprachelemente in der Fehlermeldungsliste gekennzeichnet.

ACTIVATE-FLAGGING = ANS85
Mit dieser Angabe werden sowohl veraltete Sprachelemente als auch nicht standardmäßige Spracherweiterungen in der Fehlerliste durch eine Meldung der Klasse F (Severity Code F) gekennzeichnet.

ACTIVATE-FLAGGING = FIPS(...)
Mit dieser Angabe können Sprachelemente gemäß "Federal Information Processing Standard" in der Fehlerliste durch eine Meldung der Klasse F (Severity Code F) gekennzeichnet werden.

OBSOLETE-FEATURES = NO / YES
Kennzeichnung veralteter Sprachelemente

NONSTANDARD-LANGUAGE = NO / YES
Kennzeichnung von Spracherweiterungen gegenüber dem ANS85

ABOVEMIN-SUBSET = NO / YES
Kennzeichnung aller Sprachelemente, die über die minimale Sprachmenge des ANS85 (subset minimum) hinausgehen, d.h. zur mittleren oder hohen Sprachmenge (subset intermediate oder high) gehören

ABOVEINTERMED-SUBSET = NO / YES
Kennzeichnung aller Sprachelemente, die über die mittlere Sprachmenge des ANS85 (subset intermediate) hinausgehen, d.h. zur hohen Sprachmenge (subset high) gehören

REPORT-WRITER = NO / YES
Kennzeichnung aller Sprachelemente des Listensteuerprogramms (Report Writer)

ALL-SEGMENTATION = NO / YES
Kennzeichnung aller Sprachelemente bezüglich Segmentierung

SEGMENTATION-ABOVE1 = NO / YES
Kennzeichnung aller Sprachelemente bezüglich Segmentierung auf Level 2

INTRINSIC-FUNCTIONS = NO / YES
Kennzeichnung aller Sprachelemente für interne Standardfunktionen

In den Meldungstexten werden folgende Bezeichnungen verwendet:

„obsolete“	für die veralteten Sprachelemente
„nonconforming nonstandard“	für alle Spracherweiterungen gegenüber ANS85
„nonconforming standard“	für alle Sprachelemente des ANS85, die über die eingestellte Sprachmenge (subset) hinausgehen

3.3.4 COMPILER-ACTION-Option

Diese Option legt fest, nach welchem Übersetzungschritt der Compilerlauf beendet werden soll und - falls ein Modul erzeugt wird - welches Format und welche Eigenschaften der Modul erhalten soll.

Format

```
COMPILER-ACTION = PRINT-MESSAGE-LIST / SYNTAX-CHECK / SEMANTIC-CHECK /
                                  MODULE-GENERATION(...)

  MODULE-GENERATION(...)

      ,SHAREABLE-CODE = NO / YES
      ,PREPARE-CANCEL-STMT = NO / YES
      ,MODULE-FORMAT = OM / LLM
      ,SUPPRESS-GENERATION = NO / AT-SEVERE-ERROR
      ,DESTINATION-CODE = STD / RISC-4000
      ,SEGMENTATION = ELABORATE / IGNORE
```

COMPILER-ACTION = PRINT-MESSAGE-LIST
Der Compiler gibt eine Liste aller möglichen Fehlermeldungen aus. Eine Übersetzung findet nicht statt.
Dieser Operand ist in COBOL-BC nicht verfügbar.

COMPILER-ACTION = SYNTAX-CHECK
Der Compiler prüft ein Quellprogramm nur auf syntaktische Fehler.

COMPILER-ACTION = SEMANTIC-CHECK
Der Compiler prüft die Syntax des Quellprogramms und zusätzlich die Einhaltung der semantischen Regeln. Da kein Modul erzeugt wird, können nur die Quellprogrammliste und die Fehlerliste angefordert werden.

COMPILER-ACTION = MODULE-GENERATION(...)
Es werden ein vollständiger Übersetzungslauf durchgeführt und - falls nicht explizit unterdrückt - Module erzeugt.

SHAREABLE-CODE = NO / YES
Bei Angabe von YES schreibt der Compiler den Code der PROCEDURE DIVISION (ohne DECLARATIVES) in ein gemeinsam benutzbares Codemodul (siehe 6.7).
Der Name dieses Codemoduls besteht aus dem ggf. auf 7 Zeichen gekürzten PROGRAM-ID-Namen, gefolgt von dem Zeichen "@". Jede Segmentierung der PROCEDURE DIVISION wird ignoriert.

PREPARE-CANCEL-STMT = NO / YES
Bei Angabe von YES legt der Compiler Bereiche für die Initialisierung an. Jedes Programm, auf das sich eine CANCEL-Anweisung bezieht oder das die INITIAL-Klausel enthält, sollte für einen standardkonformen Ablauf mit
PREPARE-CANCEL-STMT = YES übersetzt werden.

MODULE-FORMAT = OM / LLM
Die folgenden Angaben werden ignoriert, wenn das Modul in das POSIX-Dateiensystem geschrieben wird (siehe MODULE-OUTPUT = <c-string...>).
OM: Das Modul soll zur Weiterverarbeitung mit BINDER / TSOSLNK bzw. DBL im OM-Format (Objektmodul-Format) erzeugt werden. Maximale Länge der externen Namen: 8 Zeichen.
LLM: Das Modul soll zur Weiterverarbeitung mit dem BINDER bzw. dem DBL im LLM-Format (Bindelademodul-Format) erzeugt werden. Maximale Länge für externe Namen: 30 Zeichen.

SUPPRESS-GENERATION = NO / AT-SEVERE-ERROR
Tritt bei der Übersetzung ein Fehler mit Severity Code >= 2 auf, kann mit der Angabe AT-SEVERE-ERROR die Erzeugung des Moduls unterdrückt werden.

DESTINATION-CODE=STD / RISC-4000
Bei Angabe von STD wird /390-Code generiert.
Bei RISC-4000 wird RISC-Code für die RISC-Anlagen unter OSD-SVP generiert
Für DESTINATION-CODE=RISC-4000 ist nur das Modul-Format LLM zulässig.

SEGMENTATION=ELABORATE / IGNORE
ELABORATE: Segmentierung wird unterstützt. Wenn das Programm 'Nested Source Programs' und nicht-feste Segmente (Segment-Nummer größer oder gleich Segment-Limit) enthält, wird die Übersetzung mit einer Meldung abgebrochen. Liegt diese Kombination nicht vor, werden nur segmentierungsbezogene Sprachmittel mit entsprechenden Warnungen abgewiesen. Die Angabe SEGMENTATION = ELABORATE zusammen mit SHAREABLE-CODE = YES oder MODULE-FORMAT = LLM wird mit einer Fehlermeldung abgewiesen.
IGNORE: Segmentierungsbezogene Sprachmittel (SEGMENT-LIMIT Klausel, Segment-Nummern in 'Section-Header') werden ignoriert. Bei ihrem Auftreten wird darauf mit entsprechenden Warnungen hingewiesen.

3.3.5 MODULE-OUTPUT-Option

Mit dieser Option steuert der Benutzer, in welche Bibliothek und unter welchem Namen das Modul abgelegt werden soll.

Format

```
MODULE-OUTPUT = *STD / *OMF / <c-string 1..1024 with-low> / *LIBRARY-ELEMENT(...)
    *LIBRARY-ELEMENT(...)
        |  LIBRARY=<filename 1..54>
        |,ELEMENT = *STD (...) / <composed-name 1..32>(...)
        |    *STD (...)
        |        |  VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING /
        |        |                <alphanum-name 1..24>
        |    <composed-name>(...)
        |        |  VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING /
        |        |                <alphanum-name 1..24>
```

MODULE-OUTPUT = *STD
Ein Objektmodul wird in die temporäre EAM-Datei der aktuellen Task ausgegeben.
Ein Bindelademodul wird in eine PLAM-Bibliothek mit dem Standardnamen PLIB.COB85.<prog-id-name> ausgegeben, wobei als Elementname der Programmname verwendet wird und als Versionsbezeichnung *UPPER-LIMIT (d.h. höchstmögliche Versionsnummer) angenommen wird.

MODULE-OUTPUT = *OMF
Ein Objektmodul wird in die temporäre EAM-Datei geschrieben. Falls *OMF für ein Bindelademodul (LLM) angegeben wird, gibt der Compiler eine entsprechende Informationsmeldung aus und das Modul wird in die PLAM-Bibliothek PLIB.COB85.<prog-id-name> ausgegeben.

MODULE-OUTPUT = <c-string 1..1024 with-low>
Wenn das POSIX-Subsystem vorhanden ist, kann mit diesem Parameter das Modul (nur als LLM) als Objektdatei in das POSIX-Dateisystem geschrieben werden.
Enthält <c-string> keinen Dateiverzeichnisnamen, wird die Objektdatei unter dem angegebenen Dateinamen in das Home-Dateiverzeichnis der aktuellen BS2000-Benutzerkennung geschrieben. Soll die Objektdatei in ein anderes Dateiverzeichnis geschrieben werden, muß mit <c-string> der absolute Pfadname angegeben werden.
Bei der Namensbildung ist zu beachten, daß Objektdateien im POSIX-Subsystem nur weiterverarbeitet, d.h. gebunden werden können, wenn der Name das Suffix „.o“ enthält. Eine Namensprüfung durch den Compiler findet nicht statt.

Dieser Operand ist in COBOL-BC nicht verfügbar.

MODULE-OUTPUT = LIBRARY-ELEMENT(...)
Mit diesem Parameter wird angegeben, in welcher PLAM-Bibliothek (LIBRARY=) und unter welchem Elementnamen (ELEMENT=) das Modul abgelegt werden soll.

LIBRARY=<filename 1..54>
Name der PLAM-Bibliothek, in die das Modul geschrieben werden soll. Wenn die PLAM-Bibliothek noch nicht existiert, wird sie automatisch angelegt.

ELEMENT = *STD
Der Elementname des Moduls wird aus dem PROGRAM-ID-Namen abgeleitet.
Die Bildung der standardmäßigen Elementnamen ist in Abschnitt 2.3.1, Tabelle 2-3, dargestellt.

VERSION =
Angabe der Versionsbezeichnung

VERSION = *UPPER-LIMIT
Wird keine Versionsbezeichnung oder *UPPER-LIMIT angegeben, erhält das Element die höchstmögliche Versionsnummer (vom LMS angezeigt mit "@").

VERSION = *INCREMENT
Das Element erhält die gegenüber der höchsten vorhandenen Version um 1 inkrementierte Versionsnummer, vorausgesetzt, die höchste vorhandene Versionsbezeichnung endet mit einer inkrementierbaren Ziffer. Andernfalls ist die Versionsbezeichnung nicht inkrementierbar. In diesem Fall wird *UPPER-LIMIT angenommen und eine entsprechende Fehlermeldung ausgegeben.

Beispiel:

höchste vorhandene Version	durch *INCREMENT erzeugte Version
ABC1	ABC2
ABC	@ und Fehlermeldung
ABC9	@ und Fehlermeldung
ABC09	ABC10
003	004
keine	001

VERSION = *HIGHEST-EXISTING
Die höchste in der Bibliothek vorhandene Version wird überschrieben.

VERSION = <alphanum-name 1..24>
Das Element erhält die angegebene Versionsbezeichnung. Soll die Versionsbezeichnung inkrementierbar sein, muß mindestens das letzte Zeichen eine inkrementierbare Ziffer sein (siehe obiges Beispiel).

ELEMENT = <composed-name 1..32>
Für Bindelademodule (LLMs) kann der Benutzer einen selbstgewählten Elementnamen angeben.
Dieser Operand wird bei der Übersetzung einer Quellprogramm-Folge ignoriert. Stattdessen werden die Elementnamen der LLMs aus dem jeweiligen PROGRAM-ID-Namen abgeleitet (siehe Abschnitt 2.3.1, Tabelle 2-3).

VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING / <alphanum-name 1..24>
Versionsangabe (siehe oben: Versionsangabe für Objektmodule);
Bei der Übersetzung einer Quellprogramm-Folge erhält jedes Element die gleiche Versionsbezeichnung.

3.3.6 LISTING-Option

Die Parameter dieser Option steuern, welche Listen der Compiler erzeugen soll, welches Layout die Listen haben und wohin sie ausgegeben werden sollen.

Format

```
LISTING = NONE / STD / PARAMETERS(...)
    PARAMETERS(...)
        | OPTIONS = NO / YES
        |,SOURCE = NO / YES(...)
        |    YES(...)
        |        | COPY-EXPANSION = NO / VISIBLE-COPIES / ALL-COPIES
        |        |,SUBSCHEMA-EXPANSION = NO / YES
        |        |,INSERT-ERROR-MSG = NO / YES
        |,DIAGNOSTICS = NO / YES(...)
        |    YES(...)
        |        | MINIMAL-WEIGHT = NOTE / WARNING / ERROR / SEVERE-ERROR / FATAL-ERROR
        |        |,IMPLICIT-SCOPE-END = STD / REPORTED
        |        |,MARK-NEW-KEYWORDS = NO / YES
        |        |,REPORT-2-DIGIT-YEAR = *ACCEPT-STMT / *NO
        |,NAME-INFORMATION = NO / YES(...)
        |    YES(...)
        |        | SORTING-ORDER = ALPHABETIC / BY-DEFINITION
        |        |,CROSS-REFERENCE = NONE / REFERENCED / ALL
        |        |,SUPPRESS-GENERATION = NO / AT-SEVERE-ERROR
        |,LAYOUT = STD / PARAMETERS(...)
        |    PARAMETERS(...)
        |        | LINES-PER-PAGE = 64 / <integer 20..128>
        |        |,LINE-SIZE =  132 / <integer 119..172>
        |,OUTPUT = SYSLST / STD-FILES / LIBRARY-ELEMENT(...)
        |    LIBRARY-ELEMENT(...)
        |        | LIBRARY = <filename 1..54>
```

LISTING = NONE
Der Compiler soll keine Listen erzeugen.

LISTING = STD
Es werden die voreingestellten Werte der nachfolgenden PARAMETERS-Struktur übernommen.

LISTING = PARAMETERS(...)
Mit den folgenden Parametern wird bestimmt, welche Listen erzeugt werden und welches Layout und Ausgabeziel die angeforderten Listen haben sollen.

OPTIONS = NO / YES
Der Compiler erzeugt standardmäßig eine Liste, in der die während der Übersetzung wirksamen Steueranweisungen, die Umgebung des Übersetzungsprozesses sowie einige Informationen für Wartungs- und Diagnosezwecke aufgeführt sind.

SOURCE = YES(...)
Der Compiler erzeugt eine Quellprogrammliste und eine Bibliotheksliste.

COPY-EXPANSION = NO
Die in das Quellprogramm kopierten COPY-Elemente werden nicht in der Quellprogrammliste abgedruckt. Diese Angabe empfiehlt sich bei häufig vorkommenden COPY-Elementen, um Papier zu sparen.

COPY-EXPANSION = VISIBLE-COPIES
In der Quellprogrammliste werden nur diejenigen COPY-Elemente abgedruckt, die keine SUPPRESS-Angabe enthalten. Jede Zeile eines COPY-Elements ist in Spalte 1 der Quellprogrammliste mit einem "C" gekennzeichnet.

COPY-EXPANSION = ALL-COPIES
In der Quellprogrammliste werden alle COPY-Elemente abgedruckt, auch diejenigen, die eine SUPPRESS-Angabe enthalten. Jede Zeile eines COPY-Elements ist in Spalte 1 der Quellprogrammliste mit einem "C" gekennzeichnet.

SUBSCHEMA-EXPANSION = NO / YES
Mit der Angabe von YES wird die SUB-SCHEMA SECTION aufgelistet und jede Zeile mit einem "D" in Spalte 1 gekennzeichnet.
Dieser Operand ist in COBOL-BC nicht verfügbar.

INSERT-ERROR-MSG = NO / YES
Bei Angabe von YES werden in die Quellprogrammliste alle bei der Übersetzung aufgetretenen (Fehler-)Meldungen „eingemischt“. Die Meldungszeile steht dabei jeweils unmittelbar nach der Quellprogrammzeile, in der das meldungsauslösende Konstrukt beginnt. Meldungen, die der Compiler keiner bestimmten Quellprogrammzeile zuordnen kann, werden nach der letzten Quellprogrammzeile ausgegeben.
Der Operand wirkt auch dann, wenn keine Fehlerliste angefordert wurde.
Um ein ordnungsgemäßes Einmischen zu gewährleisten, sollte die Quellprogrammliste nicht mehr als 65535 Quellprogrammzeilen beinhalten (siehe Anhang Quellprogrammliste).

DIAGNOSTICS = YES(...)
Der Compiler erzeugt eine Fehlerliste.

MINIMAL-WEIGHT = NOTE / WARNING / ERROR / SEVERE-ERROR / FATAL-ERROR
In der Fehlerliste stehen keine Meldungen, deren Fehlergewicht kleiner ist als der angegebene Wert. Der voreingestellte Wert NOTE bewirkt, daß alle bei der Übersetzung aufgetretenen (Fehler-)Meldungen in der Liste aufgeführt werden.

IMPLICIT-SCOPE-END = STD / REPORTED
Bei Angabe von REPORTED wird in der Fehlerliste die Beendigung einer strukturierten Anweisung durch einen Punkt mit einer Hinweismeldung versehen.

MARK-NEW-KEYWORDS = NO / YES
Die Angabe von YES veranlaßt, daß Schlüsselwörter aus dem zukünftigen Standard in der Fehlerliste durch eine Meldung mit Severity-Code I gekennzeichnet werden.

REPORT-2-DIGIT-YEAR = *ACCEPT-STMT / *NO
Bei *ACCEPT-STMT bringt der Compiler für jede ACCEPT-Anweisung und für jede darin angesprochene Variable einen Hinweis, daß dort mit Jahreszahlen ohne Jahrhundert gearbeitet wird. MINIMAL-WEIGHT sollte auf NOTE stehen. Der Wert *NO unterdrückt die Ausgabe solcher Hinweise.

NAME-INFORMATION = NO / YES(...)
Bei Angabe von YES erzeugt der Compiler eine Adreßliste oder eine Adreß- und Querverweisliste. Die Liste enthält die Daten-, Kapitel- und Paragraphennamen.

SORTING-ORDER = ALPHABETIC
Die symbolischen Namen der Adreßliste sind alphabetisch aufsteigend sortiert aufgelistet.

SORTING-ORDER = BY-DEFINITION
Die symbolischen Namen der Adreßliste sind in der Reihenfolge aufgelistet, wie sie im Quellprogramm definiert wurden.

CROSS-REFERENCE = NONE
Es wird keine Querverweisliste erzeugt.

CROSS-REFERENCE = REFERENCED
Es wird eine Querverweisliste erzeugt, in der nur die Daten- und Prozedurnamen aufgelistet sind, die im Programm tatsächlich angesprochen werden.

CROSS-REFERENCE = ALL
Es wird eine Querverweisliste mit allen Daten- und Prozedurnamen erzeugt.

SUPPRESS-GENERATION = NO / AT-SEVERE-ERROR
Mit der Angabe AT-SEVERE-ERROR kann die Ausgabe der Adreß- und Querverweisliste unterbunden werden, falls bei der Übersetzung eine Fehlermeldung mit einem Severity Code >=2 auftritt.

LAYOUT = STD
Das Layout der erzeugten Listen entspricht den Standardeinstellungen der PARAMETERS-Struktur.

LAYOUT = PARAMETERS(...)
Mit den folgenden Parametern läßt sich das Layout der erzeugten Listen verändern.

LINES-PER-PAGE = 64 / <integer 20..128>
Mit diesem Parameter kann die maximale Zeilenzahl pro Seite der Protokoll-Listen festgelegt werden. Ein Seitenwechsel wird ausgeführt, wenn diese Zeilenzahl erreicht ist.

LINE-SIZE = 132 / <integer 119..172>
Dieser Parameter legt die maximale Anzahl von Zeichen fest, die pro Zeile gedruckt wird.

OUTPUT = SYSLST
Mit dieser Angabe werden die erzeugten Listen in die temporäre Systemdatei SYSLST geschrieben, von der sie automatisch nach Task-Ende (d.h. nach LOGOFF) auf den Drucker ausgegeben werden.

OUTPUT = STD-FILES
Mit dieser Angabe werden die angeforderten Listen in eigene katalogisierte Dateien ausgegeben. Die so erzeugten katalogisierten Dateien haben Standardnamen, die in der rechten Spalte der folgenden Tabelle genannt sind. *programmname* wird aus dem PROGRAM-ID-Namen abgeleitet und ggf. auf 16 Zeichen gekürzt.

Liste	**Dateiname**
Steueranweisungsliste	OPTLST.COB85.*programmname*
Quellprogrammliste/Bibliotheksliste	SRCLST.COB85.*programmname*
Adreßliste/Querverweisliste	LOCLST.COB85.*programmname*
Fehlerliste	ERRFIL.COB85.*programmname*

Dateinamen und Dateieigenschaften für diese katalogisierten Dateien sind standardmäßig vorgegeben. Der Benutzer kann aber mit Hilfe des SET-FILE-LINK-Kommandos diese Vorgaben ändern, z.B. Dateinamen eigener Wahl vergeben. Dazu muß er vor dem Aufruf des Compilers die gewünschten Eigenschaften in einem SET-FILE-LINK-Kommando mit den jeweiligen Dateikettungsnamen (Linknamen) verknüpfen, die der Compiler verwendet:

Liste	Linkname
Steueranweisungsliste	OPTLINK
Quellprogrammliste/Bibliotheksliste	SRCLINK
Adreßliste/Querverweisliste	LOCLINK
Fehlerliste	ERRLINK

Um die erzeugten Listen im POSIX-Dateisystem abzulegen, müssen sie mittels SDF-P-Variablen dem POSIX-Dateisystem zugewiesen werden. Die Standardnamen dieser Variablen lauten:

Liste	Name der SDF-P-Variablen
Steueranweisungsliste	SYSIOL-OPTLINK
Quellprogrammliste/Bibliotheksliste	SYSIOL-SRCLINK
Adreßliste/Querverweisliste	SYSIOL-LOCLINK
Fehlerliste	SYSIOL-ERRLINK

OUTPUT = LIBRARY-ELEMENT(LIBRARY = <filename 1..54>)
Die angeforderten Listen werden in die mit <filename> bezeichnete PLAM-Bibliothek ausgegeben. Jede Liste belegt ein eigenes Bibliothekselement vom Typ P mit der größtmöglichen Versionsnummer. Die Elemente erhalten folgende Standardnamen:

Liste	Elementname
Steueranweisungsliste	OPTLST.COB85.*programmname*
Quellprogrammliste/Bibliotheksliste	SRCLST.COB85.*programmname*
Adreßliste/Querverweisliste	LOCLST.COB85.*programmname*
Fehlerliste	ERRLST.COB85.*programmname*

programmname wird aus dem PROGRAM-ID-Namen abgeleitet und ggf. auf 16 Zeichen gekürzt. Nach der Übersetzung existiert ein TFT-Eintrag für den Linknamen LIBLINK, der mit dem Namen <filename> der PLAM-Bibliothek verknüpft ist.

Beispiel 3-2: Ausgabe von Listen in katalogisierte Dateien

Der Compiler soll nur eine Fehlerliste erzeugen und diese in die katalogisierte Datei FEHLER ausgeben.

```
/SET-FILE-LINK ERRLINK,FEHLER                                    (1)
/START-COBOL85-COMPILER?                                         (2)

 Angabe im Operandenfragebogen:
LISTING=PAR(OPTIONS=NO,SOURCE=NO)                                (3)
```

(1) Mit dem SET-FILE-LINK-Kommando wird der katalogisierten Datei FEHLER der Standard-Linkname ERRLINK zugeordnet.

(2) Compileraufruf im Menü-Modus

(3) Die Voreinstellung (Erzeugung von Optionen-, Quellprogramm- und Fehlerliste) wird verändert; der Compiler soll nur eine Fehlerliste erzeugen und diese standardmäßig in die katalogisierte Datei FEHLER ausgeben.

Beispiel 3-3: Ausgabe von Listen in eine PLAM-Bibliothek

Der Compiler soll alle Listen erzeugen und als Elemente in der PLAM-Bibliothek LISTLIB ablegen.

```
/START-COBOL85-COMPILER?                                         (1)

 Angabe im Operandenfragebogen:
LISTING=PAR(NAME-INFORMATION=YES(CROSS-REFERENCE=ALL),-
OUTPUT=*LIBRARY-ELEMENT(LIBRARY=LISTLIB))                        (2)
```

(1) Compileraufruf im Menü-Modus

(2) Die Voreinstellung (Ausgabe von Optionen-, Quellprogramm- und Fehlerliste) wird ergänzt; der Compiler soll zusätzlich eine Adreß- und Querverweisliste erzeugen und alle Listen in der PLAM-Bibliothek namens LISTLIB ablegen.

Beispiel 3-4: Ausgabe von Listen ins POSIX-Dateisystem

Der Compiler soll eine Quellprogramm- und eine Fehlerliste erzeugen und im POSIX-Dateisystem ablegen.

```
/DECL-VAR SYSIOL-SRCLINK,INIT='*P(xpl.srclst)',SCOPE=*TASK    (1)
/DECL-VAR SYSIOL-ERRLINK,INIT='*P(xpl.errlst)',SCOPE=*TASK    (1)
/START-COBOL85-COMPILER?                                      (2)
```

(1) Durch das Kommando DECL-VARIABLE wird die Variable mit dem gewünschten Dateinamen belegt, wobei der Dateiname ohne Pfadangaben die Ablage der Datei im Home-Dateiverzeichnis bewirkt.

(2) Compileraufruf im SDF-Menü-Modus

3.3.7 TEST-SUPPORT-Option

Diese Option steuert, ob mit der Testhilfe AID der Ablauf eines Programms getestet werden soll. Ferner können bestimmte Eigenschaften der Testhilfe AID festgelegt werden.

Diese Option ist in COBOL85-BC nicht verfügbar.

Format

```
TEST-SUPPORT = NONE / AID(...)
    AID(...)
        | STMT-REFERENCE = LINE-NUMBER / COLUMN-1-TO-6
        |,PREPARE-FOR-JUMPS = NO / YES
        |,WINDOW-DEBUG-SUPPORT = NO / YES
```

TEST-SUPPORT = NONE
Es wird keine Testhilfe angefordert. Der Compiler erzeugt lediglich ESD-Testhilfeinformationen vom Typ Übersetzungseinheit. Dabei wird dem Modul (bei segmentierten Programmen: allen Modulen) ein symbolischer Name zugeordnet, der aus den ersten 8 Zeichen des Namens im PROGRAM-ID-Paragraphen besteht. Beim Testen mit AID kann dieser Name zur Qualifikation des Quellprogramms verwendet werden.

TEST-SUPPORT = AID(...)
Dieser Parameter muß angegeben werden, wenn das Programm mit AID symbolisch überwacht werden soll. Der Compiler erzeugt dann sowohl LSD- als auch ESD-Testhilfeinformationen, so daß beim Testen mit AID symbolische Namen aus dem Quellprogramm (wie in Handbuch [8] beschrieben) verwendet werden können.
Bei segmentierten Programmen ist die Erzeugung von LSD-Informationen - und damit symbolisches Testen mit AID - nur dann möglich, wenn das Objektmodul in eine PLAM-Bibliothek ausgegeben wird.

STMT-REFERENCE = LINE-NUMBER
Die AID-Source-Referenzen werden mit Hilfe der vom Compiler erzeugten Zeilennummern gebildet.

STMT-REFERENCE = COLUMN-1-TO-6
Die AID-Source-Referenzen werden mit Hilfe der vom Anwender vergebenen Folgenummern des Quellprogramms (Spalte 1-6) gebildet.
Das Testen mit AID ist hier nur dann sinnvoll, wenn die vergebenen Folgenummern numerisch aufsteigend sortiert sind.

PREPARE-FOR-JUMPS = NO / YES
YES muß angegeben werden, wenn beim Testen mit AID

- das AID-Kommando %JUMP angewendet werden soll (siehe Handbuch [8] und Abschnitt 7.1) oder
- Testpunkte gezielt auf Paragraphen oder Kapitel gesetzt werden sollen; z.B. beim Testen von geschachtelten GO TO-Schleifen (wie sie vom COLUMBUS-Präprozessor COLCOB erzeugt werden), in denen mehrere Paragraphenüberschriften unmittelbar aufeinander oder auf eine Kapitelüberschrift folgen.

Die Verwendung dieser Funktion vergrößert das Objekt und verlängert die Programmlaufzeit.

WINDOW-DEBUG-SUPPORT = NO / YES
Bei Angabe von YES ist es möglich, das Objekt auf Quellprogrammbasis mit Hilfe der Fenstertechnik des AID-Testplatzes (AID-FE) zu testen.

3.3.8 OPTIMIZATION-Option

Mit dieser Option lassen sich die Optimierungsmaßnahmen des Compilerrs ein- und ausschalten.

Format

```
OPTIMIZATION = STD / PARAMETERS(...)
    PARAMETERS(...)
        | CALL-IDENTIFIER = *STD / *OPTIMIZE
```

OPTIMIZATION = STD
Es gilt die Voreinstellung der PARAMETERS-Struktur.

OPTIMIZATION = PARAMETERS(...)

CALL-IDENTIFIER = *STD / *OPTIMIZE
Bei Angabe von *OPTIMIZE wird die Optimierung eingeschaltet. Mehrmalige Aufrufe des gleichen Unterprogramms über CALL bezeichner werden ohne Aufruf von Systemschnittstellen abgewickelt (möglich für die ersten 100 aufgerufenen Unterprogramme).

3.3.9 RUNTIME-CHECKS-Option

Mit dieser Option werden die Prüfroutinen des Laufzeitsystems aktiviert.

Format

```
RUNTIME-CHECKS = NONE / ALL / PARAMETERS(...)
    PARAMETERS(...)
        TABLE-SUBSCRIPTS = NO / YES
       ,FUNCTION-ARGUMENTS = NO / YES
       ,PROC-ARGUMENT-NR = NO / YES
       ,RECURSIVE-CALLS = NO / YES
       ,REF-MODIFICATION = NO / YES
```

RUNTIME-CHECKS = NONE
Es werden keine Prüfroutinen des Laufzeitsystems beansprucht.

RUNTIME-CHECKS = ALL
Alle in der PARAMETERS-Struktur genannten Prüfroutinen des Laufzeitsystems werden aktiviert.

RUNTIME-CHECKS = PARAMETERS(...)

TABLE-SUBSCRIPTS = NO / YES
Ist YES angegeben überprüft das Laufzeitsystem die Einhaltung von Tabellengrenzen (sowohl bei Subskribierung als auch bei Indizierung).
Geprüft wird, ob

- Indexwerte größer als Null sind,
- Indexwerte nicht größer als die Anzahl von Elementen in den entsprechenden Dimensionen sind,
- Indexwerte nicht größer als zugehörige Werte in DEPENDING ON-Feldern sind,
- Werte in DEPENDING ON-Feldern innerhalb der Grenzen liegen, die in entsprechenden OCCURS-Klauseln definiert sind.

Das Laufzeitsystem reagiert im Fehlerfall mit der Meldung COB9144 bzw. COB9145. Das Programm bricht ab, wenn in der RUNTIME-OPTIONS-Option ERROR-REACTION = TERMINATION angegeben wurde.

FUNCTION-ARGUMENTS = NO / YES
Bei Angabe von YES werden zur Ablaufzeit die Funktionsargumente bezüglich Wertebereich, Anzahl und Länge überprüft. Treten ungültige Werte auf, wird, je nach Art des Fehlers, eine der Meldungen COB9123, COB9124, COB9125, COB9126 oder COB9127 ausgegeben; das Programm bricht ab, wenn in der RUNTIME-OPTIONS-Option ERROR-REACTION = TERMINATION angegeben wurde.

PROC-ARGUMENT-NR = NO / YES
Mit YES wird beim Aufruf eines getrennt übersetzten COBOL-Unterprogramms geprüft, ob die Anzahl der übergebenen Parameter mit der Anzahl der erwarteten Parameter übereinstimmt. Stimmt die Anzahl nicht überein, erfolgt die Meldung COB9132; das Programm bricht ab, wenn in der RUNTIME-OPTIONS-Option ERROR-REACTION = TERMINATION angegeben wurde.
Die Prüfung ist nur wirksam, wenn das aufgerufene Programm mit dieser Option und das aufrufende Programm mit einer Compilerversion ≥ 2.0 übersetzt wurde.

RECURSIVE-CALLS = NO / YES
Bei Angabe von YES wird die Aufrufhierarchie einer Programmablaufeinheit überprüft; d.h., das Laufzeitsystem prüft, ob ein getrennt übersetztes Unterprogramm rekursiv aufgerufen wird, also noch aktiv ist. Liegt ein rekursiver Aufruf vor und enthält die CALL-Anweisung keine ON EXCEPTION-Angabe, wird der Programmlauf mit der Fehlermeldung COB9157 abgebrochen.
Jedes Quellprogramm, das ein CALL und/oder CANCEL enthält, sollte mit RECURSIVE-CALLS=YES übersetzt werden.

REF-MODIFICATION = NO / YES
Die Angabe von YES bewirkt, daß das Laufzeitsystem die Einhaltung von Datenfeldgrenzen für teilfeldselektierte Bezeichner überprüft. Sind Datenfeldgrenzen nicht eingehalten, erfolgt die Fehlermeldung COB9140 und das Programm bricht ab, wenn in der RUNTIME-OPTIONS-Option ERROR-REACTION=TERMINATION angegeben wurde.

3.3.10 COMPILER-TERMINATION-Option

Mit dieser Option kann ein fehlerzahlabhängiger Abbruch des Übersetzungslaufs initiiert werden.

Format

```
COMPILER-TERMINATION = STD / PARAMETERS(...)
    PARAMETERS(...)
        | MAX-ERROR-NUMBER = NONE / <integer 1..100>
```

COMPILER-TERMINATION = STD
Es gilt die Voreinstellung der PARAMETERS-Struktur.

COMPILER-TERMINATION = PARAMETERS(...)

MAX-ERROR-NUMBER = NONE / <integer 1..100>
Mit einer Ganzzahl wird angegeben, ab welcher Fehlerzahl der Übersetzungslauf abgebrochen werden soll. Gezählt wird ab der im MINIMAL-WEIGHT-Parameter der LISTING-Option angegebenen Fehlerklasse (Standardwert: NOTE, siehe 3.3.6).

Die vorgegebene Fehlerzahl kann ggf. überschritten werden, da der Abbruch der Übersetzung immer erst nach Ablauf eines Compilersegments erfolgt (siehe Anhang).

3.3.11 MONJV-Option

Mit dieser Option kann der Benutzer eine Jobvariable zur Überwachung des Compilerlaufs einrichten.

Format

MONJV = *NONE / <filename 1..54 >

MONJV = *NONE / <filename 1..54>
Mit <filename> definiert der Benutzer eine Monitorjobvariable. Während des Übersetzungslaufs hinterlegt der Compiler in der Rückkehrcode-Anzeige dieser Jobvariablen einen Code, der über aufgetretene Fehler während des Compilerlaufs Aufschluß gibt.

3.3.12 RUNTIME-OPTIONS-Option

Die Parameter dieser Option steuern bestimmte Eigenschaften des ablauffähigen COBOL-Programms.

Format

```
RUNTIME-OPTIONS = STD / PARAMETERS(...)
    PARAMETERS(...)
        ACCEPT-STMT-INPUT = UNMODIFIED / UPPERCASE-CONVERTED
        ,FUNCTION-ERR-RETURN = UNDEFINED / STD-VALUE
        ,SORTING-ORDER = STD / BY-DIN
        ,ACCEPT-DISPLAY-ASSGN = SYSIPT-AND-SYSLST / TERMINAL
        ,ERR-MSG-WITH-LINE-NR = NO / YES
        ,ERROR-REACTION = CONTINUATION / TERMINATION
        ,ENABLE-UFS-ACCESS = NO / YES
```

RUNTIME-OPTIONS = STD
Die voreingestellten Werte der PARAMETERS-Struktur werden übernommen.

RUNTIME-OPTIONS = PARAMETERS(...)

ACCEPT-STMT-INPUT = UNMODIFIED / UPPERCASE-CONVERTED
Bei Angabe von UPPERCASE-CONVERTED werden die mittels ACCEPT-Anweisung eingegebenen Kleinbuchstaben in Großbuchstaben umgewandelt, sofern die Eingabe von der Datensichtstation erfolgt.

FUNCTION-ERR-RETURN = UNDEFINED / STD-VALUE
Bei Angabe von STD-VALUE werden die Funktionsargumente bezüglich Wertebereich, Anzahl und Länge überprüft . Falls ungültige Argumentwerte auftreten, wird der jeweiligen Funktion der entsprechende Fehler-Returnwert zugewiesen.

SORTING-ORDER = STD / BY-DIN
Die Angabe von BY-DIN veranlaßt das Dienstprogramm SORT, nach der DIN-Norm für EBCDIC zu sortieren. Dabei werden
– die Kleinbuchstaben den entsprechenden Großbuchstaben gleichgesetzt,
– die Zeichen ä/Ä mit AE, ö/Ö mit OE, ü/Ü mit UE sowie ß mit SS identifiziert
– die Ziffern vor den Buchstaben einsortiert.

ERR-MSG-WITH-LINE-NR = NO / YES
Bei Angabe von YES wird statt der Meldung COB9101 die Meldung COB9102 ausgegeben, die zusätzlich die vom Compiler vergebene Quellprogramm-Zeilennummer der Anweisung enthält, bei deren Ausführung die Meldung ausgegeben wurde.

ACCEPT-DISPLAY-ASSGN = SYSIPT-AND-SYSLST / TERMINAL
Die Angabe von TERMINAL bewirkt, daß für ACCEPT- und DISPLAY-Anweisungen ohne FROM- bzw. UPON-Angaben statt der Systemdateien SYSIPT bzw. SYSLST (Voreinstellung) die Systemdateien SYSDTA bzw. SYSOUT zugewiesen werden.

ERROR-REACTION = CONTINUATION / TERMINATION
Standardmäßig (CONTINUATION) wird der Programmablauf nach der Ausgabe folgender Meldungen fortgesetzt:
COB9120 bis COB9127, COB9131, COB9132, COB9134, COB9140, COB9144, COB9145 und COB9197.
Bei Angabe von TERMINATION führen die o.g. Fehlerfälle zu einer abnormalen Programmbeendigung (siehe auch 6.6, Programmbeendigung).

ENABLE-UFS-ACCESS = NO / YES
Bei Angabe von YES erzeugt der Compiler ein Objekt,
– das als Programm auf das POSIX-Dateisystem zugreifen kann
– im POSIX-Subsystem weiterverarbeitet (gebunden) werden kann.

Wie auf eine Datei im POSIX-Dateisystem zugegriffen wird und welchen Bedingungen die Dateiverarbeitung unterliegt, ist in Kapitel 13 beschrieben.

Dieser Operand ist in COBOL-BC nicht verfügbar.

4 Steuerung des Compilers mit COMOPT-Anweisungen

Der COBOL85-Compiler kann auch wie bisher über COMOPT-Anweisungen gesteuert werden. Er wird zu diesem Zweck mit dem Kommando

START-PROGRAM [FROM-FILE =] $COBOL85

aufgerufen.

Die Eingabe des Quellprogramms, die Ausgabe der Protokollisten und des Moduls sowie der interne Ablauf des Übersetzungsvorgangs lassen sich durch Optionen steuern, die der Benutzer in einer oder mehreren COMOPT-Anweisungen angibt. Die Optionen werden über SYSDTA nach Aufruf des COBOL85 gelesen. Der Benutzer hat drei Möglichkeiten, die Optionen einzugeben:

– Er kann die COMOPT-Anweisung(en) direkt eingeben, indem er den Compiler aufruft, ohne vorher mit dem ASSIGN-SYSDTA-Kommando die Systemdatei SYSDTA umzuweisen. Der Compiler fordert explizit die Optionen an, indem er einen Stern (*) in Spalte 1 vorgibt.

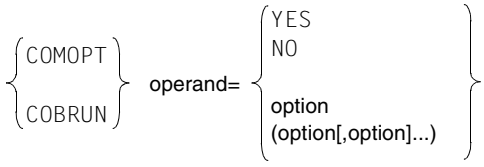
– Er kann die COMOPT-Anweisung(en) in eine Datei schreiben und über diese Datei eingeben. Diese Datei kann eine Quellprogrammdatei sein - die Optionen stehen dann vor dem Quellprogramm - oder eine eigene Datei.
Mit einem ASSIGN-SYSDTA-Kommando wird diese Datei vor dem Aufruf des Compilers der Systemdatei SYSDTA zugeordnet.

– Er kann COMOPT-Anweisungen direkt eingeben und mit der END-Anweisung SYSDTA auf eine Datei umweisen, in der vor dem Quellprogramm weitere COMOPT-Anweisungen stehen.

Wenn keine Steueranweisungen mehr erkannt werden, beginnt der Compiler sofort mit dem Lesen des Programmtextes.
Über die END-Anweisung bestimmt der Compiler den Ort des Quellprogramms und liest dort weiter.

Im Batch-Prozeß wird bei fehlerhafter Eingabe von COMOPT- oder END-Anweisungen die Übersetzung abgebrochen (Fehlermeldung COB9005).

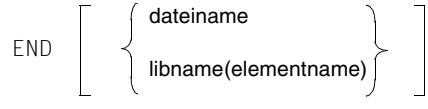
Format der COMOPT-Anweisung



– Eingabezeichen für COMOPT-Anweisungen können bis zu 128 zeichen lang sein. Bei ISAM-Dateien ist die Länge des Satzschlüssels darin enthalten. Das standardisierte Referenzformat für das Schreiben von COBOL-Quellprogrammen hat für die Eingabe von COMOPT-Anweisungen keine Bedeutung.
– Ein Operand besteht aus einem Schlüsselwort, gefolgt vom Gleichheitszeichen und einem oder mehreren Parametern. Können in einem Operanden mehrere Parameter angegeben werden, so müssen diese in Klammern gesetzt werden.

Werden bei der Abarbeitung einer COMOPT-Anweisung Fehler entdeckt, so bleiben die bereits ausgewerteten Optionen dieser Zeile in Kraft. Entsprechend der Fehlermeldung wird dann der Rest einer Operandenzeile oder der Rest des Operanden ignoriert. Fehlermeldungen für Operanden werden nur nach SYSOUT ausgegeben. Die COMOPT-Anweisungen gelten nur für den Übersetzungslauf, für den sie angegeben wurden.
Wird dieselbe COMOPT-Anweisung mehrmals angegeben, so gilt der zuletzt angegebene Wert.
Werden einander widersprechende COMOPT-Anweisungen angegeben, gilt die zuletzt angegebene Anweisung.

Format der END-Anweisung



Mit END dateiname bzw. libname kann SYSDTA auf eine Datei oder ein Bibliothekselement umgewiesen werden.

Mit END (ohne weitere Angabe) kann dem Compiler angezeigt werden, daß die Eingabe von COMOPT-Anweisungen abgeschlossen ist und die Übersetzung des Quellprogramms beginnen kann.

4.1 Quelldaten-Eingabe bei COMOPT-Steuerung

Eingaben in den Compiler können folgende Quelldaten sein:

– Quellprogramme (einzelne Quellprogramme oder Quellprogramm-Folgen)

– Quellprogrammteile (COPY-Elemente)

– COMOPT-Anweisungen

Der Compiler erwartet die Quelldaten von der System-Eingabedatei SYSDTA.

SYSDTA ist standardmäßig im Dialogbetrieb der Datensichtstation und im Stapelbetrieb der SPOOLIN- bzw. der ENTER-Datei zugewiesen.
Sollen Quelldaten direkt eingegeben werden, so ist keine Steuerung der Eingabe erforderlich. Der Compiler wird aufgerufen und die Steueranweisungen und Quellprogramm direkt eingegeben.
Kommen Quelldaten aus einer katalogisierten Datei bzw. Bibliothek, so muß die Eingabedatei SYSDTA explizit zugewiesen werden. Für die Steuerung der Eingabe von COPY-Elementen stehen eigene Steueranweisungen zur Verfügung. Die Zuweisung mit dem ASSIGN-SYSDTA-Kommando und die Eingabe von COPY-Elementen ist in Abschnitt 2.2 beschrieben.

Zuweisen des Quellprogramms mit der END-Anweisung

Die Eingabe von Quellprogrammen und Steueranweisungen kann auch ohne Verwendung des ASSIGN-SYSDTA-Kommandos erfolgen. Nach dem Aufruf erwartet der Compiler die Eingabe über SYSDTA von der Datensichtstation. Der Benutzer kann, wenn der Stern in Spalte 1 erscheint, Quellcode oder Compiler-Optionen eingeben. Alle eingegebenen Zeichen, die nicht eine gültige COMOPT-Steueranweisung darstellen, interpretiert der Compiler als Quellcode.
Mit der END-Anweisung kann eine katalogisierte Datei oder ein Bibliothekselement zugewiesen werden. Die END-Anweisung mit Angabe einer Datei oder eines Bibliothekselements kann auch die erste Anweisung nach Aufruf des Compilers sein. Am Anfang der zugewiesenen Datei können weitere COMOPT-Anweisungen stehen.

Hinweis

Falls mit der END-Anweisung ein Bibliothekselement zugewiesen wird, kann der Quellprogrammname in den Übersetzungslisten und an der Schnittstelle zu AID-FE nicht korrekt abgebildet werden.

Beispiel 4-1: Zuweisen einer katalogisierten Datei nach Eingabe von COMOPT-Anweisungen

```
/START-PROGRAM $COBOL85 ------------------------------------------------------ (1)
*COMOPT... ------------------------------------------------------------------- (2)
*END QUELL.EINXEINS ---------------------------------------------------------- (3)
```

(1) Der Compiler wird aufgerufen. SYSDTA ist im Dialog der Datensichtstation zugeordnet.

(2) Das Schlüsselwort COMOPT teilt dem Compiler mit, daß die folgenden Eingaben Steueranweisungen sind.

(3) Die END-Anweisung weist SYSDTA der katalogisierten Datei QUELL.EINXEINS zu, in der sich das zu übersetzende Quellprogramm oder eine Folge von Steueranweisungen befinden.
Am Ende der Übersetzung werden SYSDTA und SYSCMD zusammengeschaltet.

Beispiel 4-2: Zuweisen einer Bibliothek ohne COMOPT-Anweisungen

```
/START-PROGRAM $COBOL85 ------------------------------------------------------ (1)
*END PLAM.LIB(BEISP3) -------------------------------------------------------- (2)
```

(1) Aufruf des Compilers; SYSDTA ist im Dialog der Datensichtstation zugeordnet.

(2) Die Systemdatei SYSDTA wird dem Element BEISP3 in der PLAM-Bibliothek PLAM.LIB zugewiesen.
Am Ende der Übersetzung werden SYSDTA und SYSCMD zusammengeschaltet.

Zuweisen des Quellprogramms mit dem SET-FILE-LINK-Kommando und COMOPT SOURCE-ELEMENT

Die Eingabe aus Bibliotheken kann auch direkt - unter Umgehung von SYSDTA - mit dem SET-FILE-LINK-Kommando initiiert werden. Dabei muß der Standard-Linkname SRCLIB verwendet werden. Das allgemeine Format des SET-FILE-LINK-Kommandos für die Eingabe von Quellprogrammen aus Bibliotheken lautet also:

```
SET-FILE-LINK [LINK-NAME=]SRCLIB,[FILE-NAME=]libname
```

Beispiel 4-3: Eingabe aus einer PLAM-Bibliothek

```
/SET-FILE-LINK SRCLIB,PLAM.LIB ————————————————————————————— (1)
/START-PROGRAM $COBOL85 ———————————————————————————————————— (2)

*COMOPT SOURCE-ELEMENT=BEISP3 ————————————————————————————— (3)
*COMOPT SOURCE-VERSION=V001 ———————————————————————————————— (4)
*END ——————————————————————————————————————————————————————— (5)
```

(1) Mit dem SDF-Kommando (in Stellungsoperanden-Form) wird die PLAM-Bibliothek PLAM.LIB wird zugewiesen und mit dem Standard-Linknamen SRCLIB verknüpft.

(2) Aufruf des Compilers

(3) Das zu übersetzende Programm steht unter dem Elementnamen BEISP3 in der mit dem SET-FILE-LINK-Kommando zugewiesenen PLAM-Bibliothek.

(4) Die Bibliothek PLAM.LIB kann mehrere Versionen des Elements BEISP3 enthalten, von denen dasjenige mit der Versionsbezeichnung V001 eingelesen wird.

(5) Die Eingabe von Optionen ist abgeschlossen. Der Compiler beginnt mit der Übersetzung.

4.2 Tabelle der COMOPT-Operanden

Fast alle Optionen haben einen Standardwert. Gibt der Benutzer eine mögliche Option nicht explizit an, so ist der Standardwert gültig. Sollen alle vom System voreingestellten Optionen benutzt werden, so sind COMOPT-Angaben überflüssig.

Die folgende Tabelle faßt alle COMOPT-Operanden zusammen, mit denen der Compiler gesteuert werden kann.
Für die Darstellung der Anweisungsformate gilt folgendes:

– Falls ein Operand abgekürzt werden kann, steht seine Kurzform unter der ausführlichen Operandenbezeichnung (z.B. ACC-L-T-U für ACCEPT-LOW-TO-UP). Das Gleichheitszeichen zwischen Operand und Wert ist dabei in jedem Fall anzugeben.

– Voreingestellte Operandenwerte (Defaultwerte) sind im Format unterstrichen bzw. in der Kurzfassung der Funktionsbeschreibung explizit erwähnt.

In der Spalte "Funktion" ist unter dem Stichwort "SDF-Option" der zum jeweiligen COMOPT-Operanden passende SDF-Operand in Kurzform genannt. Falls kein entsprechender SDF-Operand existiert, ist dies durch einen Querstrich kenntlich gemacht.

Operandenformat	Funktion
ACCEPT-LOW-TO-UP={YES/NO} ACC-L-T-U	legt fest, ob bei der Ausführung einer ACCEPT-Anweisung eingegebene Kleinbuchstaben in Großbuchstaben umgesetzt werden sollen. Die Umsetzung erfolgt nur dann, wenn über die Datensichtstation eingegeben wird. *SDF-Option:* RUNTIME-OPTIONS = PARAMETERS(...) ACCEPT-STMT-INPUT =
ACTIVATE-WARNING-MECHANISM={YES/NO} ACT-W-MECH	gibt an, ob bei der Übersetzung im Programm enthaltene - veraltete Sprachelemente und - nicht standardgemäße Spracherweiterungen durch eine Meldung der Klasse F (Severity Code F) in der Fehlerliste gekennzeichnet werden sollen. Hinweis In Compilerläufen, für die ACTIVATE-WARNING-MECHANISM=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO Außerdem wird in diesem Fall der Operand MINIMAL-SEVERITY = I gesetzt, damit die Meldungen der Klasse F aufgelistet werden können. *SDF-Option:* ACTIVATE-FLAGGING = ANS85
ACTIVATE-XPG4-RETURNCODE={YES/NO}	bestimmt, daß nach Aufruf von C-Unterprogrammen deren Funktionswerte im COBOL-Sonderregister RETURN-CODE zur Verfügung stehen. *SDF-Option:* SOURCE-PROPERTIES = PARAMETERS(...) RETURN-CODE =
CHECK-DATE={YES/NO} CHECK-D	entscheidet, ob der Compiler bei ACCEPT FROM DATE/DAY einen Hinweis auf zweistellige Jahreszahlen ausgeben soll. *SDF-Option*: LISTING=PARAMETERS(...) DIAGNOSTICS=YES REPORT-2-DIGIT-YEAR=

Operandenformat	Funktion
CHECK-CALLING-HIERARCHY={YES/NO} CHECK-C-H	legt fest, ob die Aufrufhierarchie überprüft werden soll. Ein Programm, in dem die Anweisungen CALL bezeichner und/oder CANCEL verwendet werden, sollte mit CHECK-CALLING-HIERARCHY=YES übersetzt werden. *SDF-Option:* RUNTIME-CHECKS = PARAMETERS(...) RECURSIVE-CALLS =
CHECK-FUNCTION-ARGUMENTS={YES/NO} CHECK-FUNC	bewirkt, daß die Gültigkeit von Funktionsargumenten überprüft wird, und das Laufzeitsystem im Fehlerfall eine Meldung ausgibt. *SDF-Option:* RUNTIME-CHECKS = PARAMETERS(...) FUNCTION-ARGUMENTS =
CHECK-PARAMETER-COUNT={YES/NO} CHECK-PAR-C	bestimmt, ob beim Aufruf eines COBOL-Unterprogramms die zahlenmäßige Übereinstimmung zwischen den übergebenen und den erwarteten Parametern geprüft werden soll. *SDF-Option:* RUNTIME-CHECKS = PARAMETERS(...) PROC-ARGUMENT-NR =
CHECK-REFERENCE-MODIFICATION = {YES/NO} CHECK-REF	bestimmt, ob das Laufzeitsystem die Einhaltung von Datenfeldgrenzen für teilfeldselektierte Bezeichner überprüfen soll. *SDF-Option:* RUNTIME-CHECKS = PARAMETERS(...) REF-MODIFICATION =
CHECK-SCOPE-TERMINATORS={YES/NO} CHECK-S-T	prüft, ob die Anweisungen in der PROCEDURE DIVISION syntaktisch richtig bereichsbegrenzt sind. *SDF-Option:* LISTING = PARAMETERS(...) DIAGNOSTICS = YES(...) IMPLICIT-SCOPE-END =
CHECK-SOURCE-SEQUENCE={YES/NO} CHECK-S-SEQ	entscheidet, ob im Quellprogramm Satzpaare, die in nicht aufsteigender Reihenfolge gefunden werden, durch eine Meldung der Fehlerklasse 0 (Severity Code 0) in der Fehlerliste gekennzeichnet werden sollen. *SDF-Option:* --

Operandenformat	**Funktion**
CHECK-TABLE-ACCESS={YES/NO} CHECK-TAB	entscheidet, ob das Laufzeitsystem die Einhaltung von Tabellengrenzen überprüfen soll (sowohl für Subskribierung als auch für Indizierung). *SDF-Option:* RUNTIME-CHECKS = PARAMETERS(...) TABLE-SUBSCRIPTS =
CONTINUE-AFTER-MESSAGE={YES/NO} CON-A-MESS	entscheidet, ob das Laufzeitsystem nach Ausgabe bestimmter COB91xx-Meldungen den Programmablauf fortsetzen bzw. beenden soll. *SDF-Option:* RUNTIME-OPTIONS = PARAMETERS(...) ERROR-REACTION =
ELABORATE-SEGMENTATION={YES/NO}	Bei Angabe von NO werden segmentierungsbezogene Sprachmittel ignoriert (SEGMENT-LIMIT Klausel, Segment-Nummern in Section-Header).Es werden Warnungen ausgegeben. YES unterstützt die Segmentierung.Die Übersetzung wird aber mit Meldung abgebrochen, wenn das Programm 'Nested Source Programs' und nicht-feste Segmente enthält.Liegt diese Kombination nicht vor, werden nur segmentierungsbezogene Sprachmittel mit Warnungen abgewiesen.Auch die Angabe von ELABORATE-SEGMENTATION=YES mit GENERATE-SHARED-CODE=YES oder GENERATE-LLM=YES wird abgewiesen. *SDF-Option:* COMPILER-ACTION=MODULE-GENERATION(...) SEGMENTATION=
ENABLE-UFS-ACCESS={YES/NO} In COBOL85-BC nicht verfügbar!	legt fest, ob der Compiler ein Objekt erzeugen soll, das geeignet ist, auch Dateien aus dem POSIX-Dateisystem zu verarbeiten. *SDF-Option:* RUNTIME-OPTIONS = PARAMETERS(...) ENABLE-UFS-ACCESS =
EXPAND-COPY={YES/NO} EXP-COPY	steuert, ob in das Quellprogramm eingefügte COPY-Elemente in der Quellprogrammliste abgedruckt werden. *SDF-Option:* LISTING = PARAMETERS(...) SOURCE = YES(...) COPY-EXPANSION =

Operandenformat	Funktion
EXPAND-SUBSCHEMA={YES/NO} EXP-SUB In COBOL85-BC nicht verfügbar!	steuert, ob die SUBSCHEMA SECTION des Quellprogramms in der Quellprogrammliste protokolliert wird. *SDF-Option:* LISTING = PARAMETERS(...) SOURCE = YES(...) SUBSCHEMA-EXPANSION =
FLAG-ABOVE-MINIMUM={YES/NO}	In der Fehlerliste werden alle Sprachelemente mit F gekennzeichnet, die zur ANS85-Sprachmenge "intermediate" oder "high" gehören. Hinweis In Compilerläufen, für die FLAG-ABOVE-MINIMUM=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO *SDF-Option:* ACTIVATE-FLAGGING = FIPS(...) ABOVEMIN-SUBSET =
FLAG-ABOVE-INTERMEDIATE={YES/NO}	In der Fehlerliste werden alle Sprachelemente mit F gekennzeichnet, die zur ANS85-Sprachmenge "high" gehören. Hinweis In Compilerläufen, für die FLAG-ABOVE-INTERMEDIATE=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO *SDF-Option:* ACTIVATE-FLAGGING = FIPS(...) ABOVEINTERMED-SUBSET =

Operandenformat	Funktion
FLAG-ALL-SEGMENTATION={YES/NO}	In der Fehlerliste werden alle Sprachelemente mit F gekennzeichnet, die zur Segmentierung gehören. Hinweis In Compilerläufen, für die FLAG-ALL-SEGMENTATION=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO *SDF-Option:* ACTIVATE-FLAGGING = FIPS(...) ALL-SEGMENTATION =
FLAG-INTRINSIC-FUNCTIONS={YES/NO}	In der Fehlerliste werden alle Sprachelemente mit F gekennzeichnet, die zum Modul Intrinsic Functions gehören. Hinweis In Compilerläufen, für die FLAG-INTRINSIC-FUNCTIONS=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO *SDF-Option:* ACTIVATE-FLAGGING = FIPS(...) INTRINSIC-FUNCTIONS =
FLAG-NONSTANDARD={YES/NO}	In der Fehlerliste werden alle nicht standardgemäßen Spracherweiterungen mit F gekennzeichnet. Hinweis In Compilerläufen, für die FLAG-NONSTANDARD=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO *SDF-Option:* ACTIVATE-FLAGGING = FIPS(...) NONSTANDARD-LANGUAGE =

Operandenformat	Funktion
FLAG-OBSOLETE={YES/NO}	In der Fehlerliste werden alle veralteten Sprachelemente mit F gekennzeichnet. Hinweis In Compilerläufen, für die FLAG-OBSOLETE=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO *SDF-Option:* ACTIVATE-FLAGGING = FIPS(...) OBSOLETE-FEATURES =
FLAG-REPORT-WRITER={YES/NO}	In der Fehlerliste werden alle Sprachelemente mit F gekennzeichnet, die zum Report Writer gehören. Hinweis In Compilerläufen, für die FLAG-REPORT-WRITER=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO *SDF-Option:* ACTIVATE-FLAGGING = FIPS(...) REPORT-WRITER =
FLAG-SEGMENTATION-ABOVE1={YES/NO}	In der Fehlerliste werden alle Sprachelemente mit F gekennzeichnet, die zur Segmentierung Stufe 2 gehören. Hinweis In Compilerläufen, für die FLAG-SEGMENTATION-ABOVE1=YES angegeben wird, sind die folgenden COMOPT-Operanden unwirksam, die bei der Übersetzung eine Abweichung vom ANS85 bewirken würden: RESET-PERFORM-EXITS = NO USE-APOSTROPHE = YES REPLACE-PSEUDOTEXT = NO *SDF-Option:* ACTIVATE-FLAGGING = FIPS(...) SEGMENTATION-ABOVE1 =

Operandenformat	Funktion
GENERATE-INITIAL-STATE={YES/NO} GEN-INIT-STA	legt fest, ob der Compiler Bereiche für die Initialisierung bereitstellt. Alle Programme, die von einer CANCEL-Anweisung betroffen sind oder eine INITIAL-Klausel enthalten, sollten für einen standardkonformen Ablauf mit GENERATE-INITIAL-STATE=YES übersetzt werden. *SDF-Option:* COMPILER-ACTION = MODULE-GENERATION(...) PREPARE-CANCEL-STMT =
GENERATE-LINE-NUMBER={YES/NO} GEN-L-NUM	entscheidet, ob statt der Meldung COB9101 die Meldung COB9102 ausgegeben wird. Die Meldung COB9102 enthält zusätzlich die von COBOL85 vergebene Quellprogramm-Zeilennummer der Anweisung, deren Ausführung die jeweilige Meldung veranlaßt hat. *SDF-Option:* RUNTIME-OPTIONS = PARAMETERS(...) ERR-MSG-WITH-LINE-NR =
GENERATE-LLM={YES/NO} GEN-LLM	legt fest, mit welchem Modulformat das Modul erzeugt werden soll. Bei Angabe von YES wird ein LLM (Bindelademodul), bei Angabe von NO ein OM (Objektmodul) erzeugt. Diese Angaben werden ignoriert,falls MODUL-OUTPUT=<c-string...> angegeben wurde. *SDF-Option:* COMPILER-ACTION = MODULE-GENERATION(...) MODULE-FORMAT =
GENERATE-RISC-CODE={YES/NO} GEN-RISC	Bei NO wird /390-Code generiert. Bei YES wird RISC-Code für die RISC-Anlagen unter OSD-SVP generiert. Hierbei ist nur das Modul-Format LLM zulässig. *SDF-Option:* COMPILER-ACTION=MODULE-GENERATION(...) DESTINATION=CODE=
GENERATE-SHARED-CODE={YES/NO} GEN-SHARE	legt fest, ob der Code der PROCEDURE DIVISION (ohne DECLARATIVES) in ein eigenes Codemodul geschrieben wird. Als Name des Moduls wird der ggf. auf 7 Zeichen gekürzte Programmname mit angehängtem "@" verwendet. *SDF-Option:* COMPILER-ACTION = MODULE-GENERATION(...) SHAREABLE-CODE =

Operandenformat	Funktion
IGNORE-COPY-SUPPRESS={YES/NO} IGN-C-SUP	bestimmt, ob im Quellprogramm vorhandene COPY-Elemente mit SUPPRESS-Angabe trotzdem in der Quellprogrammliste aufgeführt werden. *SDF-Option:* LISTING = PARAMETERS(...) SOURCE = YES(...) COPY-EXPANSION =
INHIBIT-BAD-SIGN-PROPAGATION={YES/NO}	NO ermöglicht beim Übertragen eines Datenfeldes in ein anderes, die beide numerisch und mit USAGE DISPLAY beschrieben sind, das Generieren von schnellerem Code. Es wird dabei kein Code erzeugt der verhindert, daß Vorzeichenverschlüsselungen weitergegeben werden, die nicht mit der PICTURE-Klausel übereinstimmen.

Operandenformat	Funktion
LIBFILES= (listenangabe[,listenangabe]...)	legt fest, welche Übersetzungsprotokolle erzeugt und in eine PLAM-Bibliothek ausgegeben werden sollen. listenangabe ist dabei eine der folgenden Angaben: [NO]OPTIONS [NO]DIAG SYSLIST [NO]SOURCE [NO]OBJECT ALL [NO]MAP [NO]XREF NO SYSLIST beinhaltet diejenigen Listen, die bei der COMOPT SYSLIST spezifiziert worden sind. Die angeforderten Listen werden von links nach rechts abgearbeitet. Es gilt der zuletzt gesetzte Wert für die jeweilige Liste. Wird XREF angegeben, so wird automatisch auch MAP angenommen. Jede angeforderte Liste wird als Element vom Typ P mit einem Standardnamen generiert. Die Standardnamen lauten: OPTLST.COB85.programmname (Steueranweisungsliste) SRCLST.COB85.programmname (Quellprogrammliste) ERRLST.COB85.programmname (Fehlerliste) LOCLST.COB85.programmname (Adreß-/Querverweisliste) OBJLST.COB85.programmname (Objektliste) programmname wird dabei ggf. auf 16 Zeichen gekürzt. Die PLAM-Bibliothek muß mit dem SET-FILE-LINK Kommando über denLinknamen LIBLINK zugewiesen werden. Wird kein Bibliotheksname zugewiesen, legt der Compiler die angeforderten Listen in der Standard-Bibliothek PLIB.COB85.programmname ab. *SDF-Option:* LISTING = PARAMETERS(...) OUTPUT = LIBRARY-ELEMENT(...) LIBRARY=<filename 1..54>
LINE-LENGTH=132 / 119..172 LINE-L	legt die maximale Anzahl von Zeichen fest, die in den Übersetzungsprotokollen pro Zeile gedruckt werden. *SDF-Option:* LISTING = PARAMETERS(...) LAYOUT = PARAMETERS(...) LINE-SIZE =

Operandenformat	Funktion
LINES-PER-PAGE=64 / 20..128 LINES	legt die maximale Anzahl von Zeilen fest, die in den Übersetzungsprotokollen pro Seite gedruckt werden. Ein Seitenwechsel wird ausgeführt, wenn diese Zeilenzahl erreicht ist. *SDF-Option:* LISTING = PARAMETERS(...) LAYOUT = PARAMETERS(...) LINES-PER-PAGE =
LISTFILES= (listenangabe[,listenangabe]...))	legt fest, welche Übersetzungsprotokolle erzeugt und in katalogisierte Dateien ausgegeben werden sollen. listenangabe ist dabei eine der folgenden Angaben: [NO]OPTIONS [NO]DIAG SYSLIST [NO]SOURCE [NO]OBJECT ALL [NO]MAP [NO]XREF NO Weitere Beschreibung analog der COMOPT LIBFILES. *SDF-Option:* LISTING = PARAMETERS(...) OUTPUT = STD-FILES
MARK-NEW-KEYWORDS={YES/NO} M-N-K	veranlaßt, daß Schlüsselwörter aus dem zukünftigen Standard in der Fehlerliste durch eine Meldung mit Severity Code I gekennzeichnet werden. *SDF-Option:* LISTING=PARAMETERS(...) DIAGNOSTICS=YES MARK-NEW-KEYWORDS=
MAXIMUM-ERROR-NUMBER=ganzzahl MAX-ERR	legt fest, ab welcher Fehlerzahl (in Abhängigkeit von der MINIMAL-SEVERITY-Angabe) die Übersetzung abgebrochen werden soll. *SDF-Option:* COMPILER-TERMINATION = PARAMETERS(...) MAX-ERROR-NUMBER =
MERGE-DIAGNOSTICS={YES/NO} M-DIAG	bewirkt, daß alle während der Übersetzung aufgetretenen Fehlermeldungen in die Quellprogrammliste „eingemischt“ werden. Um ein ordnungsgemäßes Einmischen zu gewährleisten, sollte die Quellprogrammliste nicht mehr als 65535 Quellprogrammzeilen beinhalten (siehe Anhang, Quellprogrammliste). *SDF-Option:* LISTING=PARAMETERS(...) SOURCE=YES(...) INSERT-ERROR-MSG=

Operandenformat	Funktion
MINIMAL-SEVERITY={ I / 0 / 1 / 2 / 3 } MIN-SEV	unterdrückt in der Fehlerliste Meldungen, deren Severity Code kleiner als der angegebene Wert ist. *SDF-Option:* LISTING = PARAMETERS(...) DIAGNOSTICS = YES(...) MINIMAL-WEIGHT =
MODULE={*OMF / libname}	vereinbart, wohin das bei der Übersetzung erzeugte Objektmodul ausgegeben werden soll. *OMF bewirkt die Ausgabe in die temporäre EAM-Datei der aktuellen Task. libname ist der Name der PLAM-Bibliothek, in die das Objektmodul ausgegeben werden soll. libname muß ein zulässiger Dateiname des BS2000 sein. *SDF-Option:* MODULE-OUTPUT = *OMF / *LIBRARY-ELEMENT(...) LIBRARY =
MODULE-ELEMENT=elementname MODULE-ELEM	Angabe des Elementnamens, unter dem ein Bindelademodul (LLM) in die PLAM-Bibliothek geschrieben werden soll. Maximale Länge von elementname: 32 Zeichen Bei Objektmodulen und Quellprogrammfolgen wird diese Compileroption ignoriert und mit einer Hinweismeldung quittiert. *SDF-Option:* MODULE-OUTPUT = *LIBRARY-ELEMENT(...) LIBRARY = <filename> ,ELEMENT =
MODULE-VERSION=version MODULE-VERS	ermöglicht es, dem Element, das das bei der Übersetzung erzeugte Modul enthält, eine Versionbezeichnung zu geben. version ist eine der folgenden Angaben: *UPPER-LIMIT / *UPPER *HIGHEST-EXISTING / *HIGH *INCREMENT / *INCR <alphanum-name 1..24> Beschreibung der Angaben siehe *SDF-Option:* MODULE-OUTPUT = *LIBRARY-ELEMENT(...) LIBRARY = <filename> ,ELEMENT = <composed-name> VERSION =

Operandenformat	Funktion
OPTIMIZE-CALL-IDENTIFIER={YES/NO} O-C-I	bewirkt, daß mehrmalige Aufrufe des gleichen Unterprogramms über CALL bezeichner ohne Aufruf von Systemschnittstellen abgewickelt werden (möglich für die ersten 100 aufgerufenen Unterprogramme) *SDF-Option:* OPTIMIZATION = PARAMETERS(...) CALL-IDENTIFIER =
PRINT-DIAGNOSTIC-MESSAGES={YES/NO} PRI-DIAG In COBOL85-BC nicht verfügbar!	ermöglicht es, sich eine Liste aller COBOL85-Fehlermeldungen ausgeben zu lassen. Eine Übersetzung wird in diesem Fall nicht durchgeführt. *SDF-Option:* COMPILER-ACTION = PRINT-MESSAGE-LIST
REDIRECT-ACCEPT-DISPLAY={YES/NO}	bewirkt, daß für ACCEPT- und DISPLAY-Anweisungen ohne FROM- bzw. UPON-Angaben statt der Systemdateien SYSIPT / SYSLST (Voreinstellung) die Systemdateien SYSDTA bzw. SYSOUT zugewiesen werden. *SDF-Option:* RUNTIME-OPTIONS = PARAMETERS(...) ACCEPT-DISPLAY-ASSGN =
REPLACE-PSEUDOTEXT={YES/NO} REP-PSEUDO	entscheidet, wie COPY-Elemente in einzeln ersetzbare Textwörter zerlegt werden. Bei Angabe von NO wirken die Trennsymbole Doppelpunkt, Klammer auf, Klammer zu und Pseudotext-Begrenzer nicht als Trennzeichen für Textwörter und sind keine eigenständigen Textwörter. Das bewirkt insbesondere, daß innerhalb von Klammern in Maskenzeichenfolgen keine Ersetzungen stattfinden. Bei Angabe von NO ist eine Verwendung der REPLACE-Anweisung nicht möglich. Die Möglichkeiten in der REPLACING-Klausel sind auf die Ersetzung eines einzelnen Textwortes durch ein Textwort oder einen Bezeichner beschränkt. COPY-Elemente dürfen selbst keine COPY-Anweisungen enthalten. *SDF-Option:* --
RESET-PERFORM-EXITS={YES/NO} RES-PERF	legt fest, ob die Steuerungsmechanismen für alle PERFORM-Anweisungen - bei EXIT PROGRAM entsprechend ANS85 zurückgesetzt werden (Defaultwert oder Angabe YES) oder - beim Verlassen des Unterprogramm aktiv bleiben (Angabe NO). *SDF-Option:* --

Operandenformat	Funktion
ROUND-FLOAT-RESULTS-DECIMAL={YES/NO} ROUND-FLOAT	legt fest, ob Gleitpunktdatenfelder vor der Übertragung in Festpunktdatenfelder auf die 7. (COMP-1) bzw. 15. Dezimalstelle (COMP-2) gerundet werden sollen. *SDF-Option:* --
SEPARATE-TESTPOINTS={YES/NO} SEP-TESTP In COBOL85-BC nicht verfügbar!	bestimmt, ob zum Testen mit AID für jeden Paragraphen- und Kapitelanfang in der PROCEDURE DIVISION eine eigene Adresse generiert werden soll. *SDF-Option:* TEST-SUPPORT = AID(...) PREPARE-FOR-JUMPS =
SET-FUNCTION-ERROR-DEFAULT={YES/NO} S-F-E-D	bewirkt, daß die Gültigkeit von Funktionsargumenten überprüft wird und im Fehlerfall der jeweiligen Funktion der Fehler-Returnwert zugewiesen wird. *SDF-Option:* RUNTIME-OPTIONS = PARAMETERS(...) FUNCTION-ERR-RETURN =
SHORTEN-OBJECT={YES/NO} SHORT-OBJ In COBOL85-BC nicht verfügbar!	legt fest, ob in der angeforderten Objektliste nur die ESD-Informationen aufgeführt werden sollen. *SDF-Option:* --
SHORTEN-XREF={YES/NO} SHORT-XREF	entscheidet, ob in der gewünschten Querverweisliste nur Daten- bzw. Prozedurnamen aufgelistet werden sollen, die im Programm angesprochen werden. *SDF-Option:* LISTING = PARAMETERS(...) NAME-INFORMATION = YES(...) CROSS-REFERENCE =
SORT-EBCDIC-DIN={YES/NO} SORT-E-D	erlaubt es, für SORT das Format ED zu wählen (siehe [6]): Dadurch werden (u.a.) beim Sortieren die Umlaute ä, ö bzw. ü wie AE, OE bzw. UE behandelt. *SDF-Option:* RUNTIME-OPTIONS = PARAMETERS(...) SORTING-ORDER =

Operandenformat	Funktion
SORT-MAP={YES/NO}	gestattet es, sich die Adreßliste (LOCATOR MAP) aufsteigend sortiert nach symbolischen Namen aus dem Quellprogramm ausgeben zu lassen. Das Protokoll besteht aus Listen für Daten-, Kapitel- und Paragraphennamen. *SDF-Option:* LISTING = PARAMETERS(...) NAME-INFORMATION = YES(...) SORTING-ORDER =
SUPPORT-WINDOW-DEBUGGING = {YES/NO} S-W-D In COBOL85-BC nicht verfügbar!	ermöglicht es, das generierte Objekt im Rahmen des graphischen AID-Testplatzes (AID-FE) zu testen. Bei Angabe von SUPPORT-WINDOW-DEBUGGING=YES wird auch SEPARATE-TESTPOINT=YES angenommen. *SDF-Option:* TEST-SUPPORT = AID(...) WINDOW-DEBUG-SUPPORT =
SOURCE-ELEMENT=element SOURCE-ELEM	weist dem Compiler als Quellprogramm ein Element einer PLAM-Bibliothek zu. Vor der Übersetzung muß diese Bibliothek mit dem SET-FILE-LINK-Kommando über den Linknamen SRCLIB zugewiesen werden. element ist dabei der Name des Bibliothekselementes. Es muß in einer PLAM-Bibliothek unter dem Elementtyp S enthalten sein. element darf höchstens 40 Zeichen lang sein. *SDF-Option:* SOURCE = *LIBRARY-ELEMENT(...) LIBRARY = ELEMENT =
SOURCE-VERSION=version SOURCE-VERS	gibt dem Compiler an, welche Version des mit SOURCE-ELEMENT zugewiesenen Elementes zu übersetzen ist. version ist eine der folgenden Angaben: *HIGHEST-EXISTING / *HIGH *UPPER-LIMIT / *UPPER <alphanum-name 1..24> Beschreibung der Angaben siehe *SDF-Option:* SOURCE = *LIBRARY-ELEMENT(...) LIBRARY = ,ELEMENT = VERSION =

Operandenformat	Funktion
SUPPRESS-LISTINGS={YES/NO} SUP-LIST	ermöglicht es, beim Auftreten einer Fehlermeldung mit einem Severity-Code >= 2 die Ausgabe der - Objekt-, - Adreß- und - Querverweis-Liste zu verhindern. Ausgegeben werden dann nur (falls angefordert) die Fehlerliste und die Quellprogrammliste. *SDF-Option:* LISTING = PARAMETERS(...) NAME-INFORMATION = SUPPRESS-GENERATION =
SUPPRESS-MODULE={YES/NO} SUP-MOD	ermöglicht es, beim Auftreten einer Fehlermeldung mit einem Severity-Code >=2 die Erzeugung eines Moduls zu verhindern. SUPPRESS-MODULE=YES hat darüberhinaus den Operanden SUPPRESS-LISTINGS=YES zur Folge. *SDF-Option:* COMPILER-ACTION = MODULE-GENERATION(...) SUPPRESS-GENERATION =
SYMTEST={ALL/NO} In COBOL85-BC nicht verfügbar!	legt die Information fest, die der Compiler für die Dialogtesthilfe AID (siehe [8]) bereitstellt. ALL: Der Compiler erzeugt LSD-Informationen und ESD-Testhilfe-Informationen NO: Der Compiler erzeugt nur ESD-Testhilfe-Informationen. *SDF-Option:* TEST-SUPPORT = AID(...)
SYSLIST= (listenangabe[,listenangabe]...)	legt fest, welche Übersetzungsprotokolle erzeugt und in die Systemdatei SYSLST ausgegeben werden sollen. listenangabe ist dabei eine der folgenden Angaben: [NO]OPTIONS ALL [NO]SOURCE NO [NO]MAP [NO]DIAG [NO]OBJECT [NO]XREF *SDF-Option:* LISTING = PARAMETERS(...) OUTPUT = SYSLST

Operandenformat	Funktion
TCBENTRY=name In COBOL85-BC nicht verfügbar!	vereinbart den Namen der Verbindungstabelle zu UTM, die COBOL85 erzeugt. Diese Tabelle enthält Zeiger zu internen Arbeitsbereichen, die bei Wiederdurchlauf eines UTM Teilprogrammes von UTM erneut initialisiert werden müssen. name ist dabei der Name der Verbindungstabelle. Er kennzeichnet den Anfang dieser Zeigertabellen. name darf höchstens 8 Zeichen lang sein *SDF-Option:* --
TERMINATE-AFTER-SEMANTIC={YES/NO} TERM-A-SEM	ermöglicht es, das Quellprogramm nur auf syntaktische und semantische Fehler überprüfen zu lassen, ohne daß ein Modul erzeugt wird. Dabei können nur die Quellprogramm- und die Fehlerliste ausgegeben werden. *SDF-Option:* COMPILER-ACTION = SEMANTIC-CHECK
TERMINATE-AFTER-SYNTAX={YES/NO} TERM-A-SYN	ermöglicht es, das Quellprogramm nur auf syntaktische Fehler überprüfen zulassen, ohne daß ein Modul erzeugt wird. Dabei können nur die Quellprogramm- und die Fehlerliste ausgegeben werden. *SDF-Option:* COMPILER-ACTION = SYNTAX-CHECK
TEST-WITH-COLUMN1={YES/NO} TEST-W-C In COBOL85-BC nicht verfügbar!	legt fest, ob bei SYMTEST=ALL die AID-Source-Referenzen mit Hilfe der Folgenummern des Quellprogramms (Spalte 1 bis 6) gebildet werden sollen. *SDF-Option:* TEST-SUPPORT = AID(...) STMT-REFERENCE =
USE-APOSTROPHE={YES/NO} USE-AP	ermöglicht es, anstelle der Anführungszeichen (") im Quellprogramm Apostrophe (') als Literalbegrenzer zu vereinbaren. Diese Vereinbarung gilt auch für die figurative Konstante QUOTE. Hinweis: Ein Quellprogramm mit Apostroph (') als Literalbegrenzer entspricht nicht dem ANS85. *SDF-Option:* --

5 COBOL85-Strukturierer

In COBOL85-BC und in POSIX nicht unterstützt !

Der COBOL85-Strukturierer ist ein Programmpaket zur Aufbereitung von COBOL-Quellprogrammen. Es kann nach dem Editieren, beim Test und bei der Programmdokumentation eingesetzt werden.

Der COBOL85-Strukturierer wird über SDF gesteuert und folgendermaßen aufgerufen:

```
/START-COBOL85-STRUCTURIZER [optionen]
(Abkürzung: S-COB-S)
```

Die Bearbeitung wird sofort nach Eingabe des Kommandos gestartet.

Soll das Quellprogramm ohne explizite Eingabe von Optionen (d.h. mit den voreingestellten Operandenwerten) bearbeitet werden, muß es vor Aufruf des Strukturierers der Systemdatei SYSDTA zugewiesen worden sein.
Die SDF-Steuerung des Strukturierers entspricht formal der des Compilers; die verschiedenen Möglichkeiten der Optioneneingabe sind im Abschnitt 3.1 beschrieben und gelten auch für den Strukturierer.

Für die Eingabe von COPY-Elementen in den Strukturierer gelten dieselben Bedingungen wie für die Eingabe von COPY-Elementen in den Compiler (siehe Abschnitt 2.2.2). Insbesondere müssen ggf. die Standard-Linknamen COBLIB, COBLIB1 bis COBLIB9 verwendet werden.

Der COBOL85-Strukturierer bietet folgende Funktionen:

- Quelltext-Aufbereitung ("Beautify"-Funktion)
  - Einrücken der Quellprogrammzeilen in Übereinstimmung mit dem COBOL-Referenzformat
  - Einarbeitung expliziter Bereichsbegrenzer (Scope Terminators)
  - Umwandlung obsoleter Sprachmittel in Kommentare

  Das so aufbereitete Quellprogramm kann übersetzt werden.

  Das Ergebnis einer Quelltext-Aufbereitung wird standardmäßig in einer katalogisierten SAM-Datei namens *dateiname*.IND bzw. *elementname*.IND abgelegt.

- Strukturliste vom Quellprogramm ("Pretty-Print"-Funktion)

  Der Strukturierer erzeugt eine Quellprogramm-Liste, in der der Quelltext in aufbereiteter Form (analog zur "Beautify"-Funktion) mit graphischer Kennzeichnung der Anweisungsstruktur und - wahlweise - mit Querverweisen (Cross References) abgebildet ist.

  Die Strukturliste kann wegen der graphischen Aufbereitung nicht übersetzt werden.

  Die Strukturliste wird standardmäßig in einer katalogisierten SAM-Datei namens STRLST.COB85.*program-id-name* abgelegt.

Sprachumfang

Der Strukturierer verarbeitet alle Sprachmittel des COBOL85-Compilers im BS2000.
In der "Pretty Print" Funktion werden jedoch die Anweisungen
COPY textname REPLACING ==pseudotext==BY==pseudotext== und
REPLACE==pseudotext==BY==pseudotext==
nur syntaktisch akzeptiert; die beabsichtigte Textersetzung wird im Programmtext jedoch nicht dargestellt. COPY REPLACING word BY word wird jedoch intern verarbeitet , um ggf. ein korrektes XREF Listing zu produzieren. REPLACE- Anweisung und REPLACING-Klauseln mit Verwendung von Pseudotext oder Bezeichnern werden auch bei der XREF Erstellung nicht berüchsichtigt.

Allgemein gilt, daß COBOL Quelltext im COBOL Referenzformat geschrieben sein muß. Ein fehlerhaftes Zeichen in Spalte 7 wird bei der Ausgabe durch Zwischenraum ersetzt.

5.1 Quelltext-Aufbereitung ("Beautify"-Funktion)

Es ist empfehlenswert, das Quellprogramm vor dem Aufbereiten fehlerfrei zu editieren (z.B. durch Übersetzung mit COMPILER-ACTION=SEMANTIC-CHECK). Der Strukturierer erkennt zwar Syntaxfehler, i.A. jedoch nur einen Fehler pro Aufruf.

Nachfolgend sind die einzelnen Aufbereitungsmaßnahmen zusammengefaßt.

- Die COPY-Anweisungen im Quellprogramm werden wahlweise aufgelöst, der Text der COPY-Elemente wird nicht ausgegeben.
- Dateneinträge in der Data Division werden bei eingestellter Option DATA-FORMATTING = YES wie folgt ausgegeben:
  - Die Stufennummern 01 und 77 werden in den Spalten 8 und 9 ausgegeben.
  - Alle übrigen Stufennummern werden um je vier Spalten gegenüber der nächsthöheren Stufennummer eingerückt, jedoch höchstens bis zur Spalte 36.
  - Die Datennamen werden um vier Stellen gegenüber der zugehörigen Stufennummer eingerückt.
- Anweisungen in der Procedure Division werden wie folgt ausgegeben:
  - Jede Anweisung beginnt in einer neuen Zeile.
  - Alle IF- und EVALUATE-Anweisungen werden mit dem zugehörigen Scope Terminator abgeschlossen.
  - Das optionale Schlüsselwort THEN wird ggf. eingefügt.
  - Jede Anweisung wird gegenüber der nach der Schachtelung nächst höheren Anweisung eingerückt. Die Einrücktiefe kann mit dem INDENTATION-AMOUNT-Operanden des LAYOUT-Parameters festgelegt werden.
  - Die vom Strukturierer ergänzten Schlüsselwörter werden wahlweise kleingeschrieben.
  - Die Bedingung nach einer IF-Anweisung wird stets um drei Stellen eingerückt.
  - Die folgenden Schlüsselwörter und Ausnahmebedingungen stehen für sich in einer Zeile:

    THEN, ELSE, END-IF, WHEN OTHER, END-EVALUATE, END-PERFORM, EXIT PERFORM, [NOT] AT END, [NOT] INVALID KEY, [NOT] ON SIZE ERROR, [NOT] ON OVERFLOW, [NOT] AT END-OF-PAGE
    sowie alle expliziten Bereichsbegrenzer nach einer Ausnahmebedingung
  - Paragraphen-Namen stehen ebenfalls allein in einer Zeile.
  - Leerzeilen, Kommentarzeilen, Fortsetzungszeilen und DEBUG-Zeilen werden unverändert übernommen.

– Das Layout der Anweisungen bleibt erhalten.

– Überflüssige Punkte werden entfernt.

– Ein Punkt nach einer Ausnahmebedingung wird durch den entsprechenden expliziten Bereichsbegrenzer der Anweisung (z.B. END-READ) ersetzt. Ist die letzte Anweisung, die von der Ausnahmebedingung abhängt, dieselbe wie die bedingende Anweisung, so wird der Bereichsbegrenzer verdoppelt.

  Beispiel:

  Der Programmausschnitt

```
ADD 1 TO Z
ON SIZE ERROR
MOVE X TO FLAG
ADD 1 TO ERRN.
```

  sieht aufbereitet so aus:

```
ADD 1 TO Z
ON SIZE ERROR
  MOVE X TO FLAG
  ADD 1 TO ERRN END-ADD
END-ADD
```

  Würde nur ein END-ADD eingefügt, dann wäre nur die innere ADD-Anweisung begrenzt.

– Folgende veraltete Sprachmittel werden in Kommentarzeilen umgewandelt:
  NOTE, REMARKS, ENTER COBOL, ENTER LINKAGE, EXHIBIT, ON, READY TRACE, RESET TRACE, EXAMINE, TRANSFORM
  Die beiden letzten Anweisungen lösen zusätzlich die Aufforderung aus, sie manuell umzusetzen.

Beispiel 5-1: Quelltext-Aufbereitung

Programmausschnitt des noch nicht aufbereiteten Quellprogramms:

```
...

 PROCEDURE DIVISION.
 MAIN.
    DISPLAY "PLEASE, GIVE TRIANGLE SIDES, EACH ONE DIGIT"
                         UPON T
     PERFORM 4 TIMES
     CALL "EINGABE" USING I J K
     IF I + J NOT GREATER K OR
     J + K NOT GREATER I OR
     K + I NOT GREATER J
     THEN
     DISPLAY "NOT A TRIANGLE" UPON T
     ELSE
     MOVE ZERO TO MATCH
     IF I = J
      THEN
     ADD 1 TO MATCH
     END-IF
     IF J = K
     THEN
        ADD 1 TO MATCH
     END-IF
      IF K = I
     THEN
     ADD 1 TO MATCH
     END-IF
         EVALUATE TRUE
         WHEN MATCH = ZERO
        DISPLAY "SCALENE TRIANGLE" UPON T
         WHEN MATCH = 1
        DISPLAY "ISOSCELES TRIANGLE" UPON T
       WHEN OTHER
        DISPLAY "EQUILATERAL TRIANGLE" UPON T
         END-EVALUATE
      END-IF
      END-PERFORM
       STOP RUN.
```

Aufbereitetes Quellprogramm:

```
...

 PROCEDURE DIVISION.
 MAIN.
     DISPLAY "PLEASE, GIVE TRIANGLE SIDES, EACH ONE DIGIT"
       UPON T
     PERFORM 4 TIMES
       CALL "EINGABE" USING I J K
       IF I + J NOT GREATER K OR
          J + K NOT GREATER I OR
          K + I NOT GREATER J
       THEN
         DISPLAY "NOT A TRIANGLE" UPON T
       ELSE
         MOVE ZERO TO MATCH
         IF I = J
         THEN
           ADD 1 TO MATCH
         END-IF
         IF J = K
         THEN
           ADD 1 TO MATCH
         END-IF
         IF K = I
         THEN
           ADD 1 TO MATCH
           END-IF
         EVALUATE TRUE
         WHEN MATCH = ZERO
           DISPLAY "SCALENE TRIANGLE" UPON T
         WHEN MATCH = 1
           DISPLAY "ISOSCELES TRIANGLE" UPON T
         WHEN OTHER
           DISPLAY "EQUILATERAL TRIANGLE" UPON T
         END-EVALUATE
       END-IF
     END-PERFORM
     STOP RUN.
```

5.2 Strukturliste vom Quellprogramm ("Pretty-Print"-Funktion)

Die Strukturliste vom Quellprogramm kann in zwei Stufen erzeugt werden:

- Strukturliste ohne Querverweisangaben
- Strukturliste mit Querverweisangaben

Beiden Listen gemeinsam sind folgende Strukturierungsmöglichkeiten:

– die Quelltext-Aufbereitung analog zur "Beautify"-Funktion (siehe 5.1). Dabei wird die Einstellung DATA-FORMATTING = YES implizit angenommen, d.h. auch in der DATA DIVISION wird formatiert.

– Die COPY-Anweisungen im Quellprogramm werden wahlweise aufgelöst und der Text der COPY-Elemente ausgegeben.

– Einrahmung der Anweisungsblöcke mit waagrechten und senkrechten Linien

– Seitennumerierung

– Zeilennumerierung analog zur Zeilennumerierung der Compiler-Quellprogrammliste

– Fettdruck der vom Strukturierer in der Aufbereitung ergänzten Schlüsselwörter

– Gestaltung des Listen-Layouts (Zeilenanzahl und -länge)

Mit Ausnahme der Aufbereitung des Quelltexts können alle Strukturierungsmaßnahmen ein- bzw. ausgeschaltet werden.

5.2.1 Strukturliste ohne Querverweise

Beispiel 5-2

(erzeugt mit CROSS-REFERENCE=NO,
INFORMATION-Parameter SEQUENCE-AREA=NO,
IDENTIFICATION-AREA=NO sowie LAYOUT-Parameter LINE-SIZE=80)

```
COBOL85 V02.1B00  PRETTY-PRINTING  DATE: 1994-03-03 TIME: 10:21:08 PAGE: 1
LINE ID   SOURCE: TRIANGLE

          ...

14        PROCEDURE DIVISION.
15        MAIN.
16            DISPLAY "PLEASE, GIVE TRIANGLE SIDES, EACH ONE DIGIT"
17              UPON T
18          ┌─PERFORM 4 TIMES
19          │   CALL "EINGABE" USING I J K
20          │  ┌─IF I + J NOT GREATER K OR
21          │  │    J + K NOT GREATER I OR
22          │  │    K + I NOT GREATER J
23          │  ├─THEN──────────────────────────────────────────────────────
24          │  │   DISPLAY "NOT A TRIANGLE" UPON T
25          │  ├─ELSE──────────────────────────────────────────────────────
26          │  │   MOVE ZERO TO MATCH
27          │  │  ┌─IF I = J
28          │  │  ├─THEN──────────────────────────────────────────────────
29          │  │  │   ADD 1 TO MATCH
30          │  │  └─END-IF────────────────────────────────────────────────
31          │  │  ┌─IF J = K
32          │  │  ├─THEN──────────────────────────────────────────────────
33          │  │  │   ADD 1 TO MATCH
34          │  │  └─END-IF────────────────────────────────────────────────
35          │  │  ┌─IF K = I
36          │  │  ├─THEN──────────────────────────────────────────────────
37          │  │  │   ADD 1 TO MATCH
38          │  │  └─END-IF────────────────────────────────────────────────
39          │  │  ┌─EVALUATE TRUE
40          │  │  ├─WHEN MATCH = ZERO─────────────────────────────────────
41          │  │  │   DISPLAY "SCALENE TRIANGLE" UPON T
42          │  │  ├─WHEN MATCH = 1────────────────────────────────────────
43          │  │  │   DISPLAY "ISOSCELES TRIANGLE" UPON T
44          │  │  ├─WHEN OTHER────────────────────────────────────────────
45          │  │  │   DISPLAY "EQUILATERAL TRIANGLE" UPON T
46          │  │  └─END-EVALUATE─────────────────────────────────────────
47          │  └─END-IF──────────────────────────────────────────────────
48          └─END-PERFORM──────────────────────────────────────────────────
49        <──STOP RUN.
```

5.2.2 Strukturliste mit Querverweisen

Die Strukturliste mit Querverweisen enthält zusätzlich zur graphischen Kennzeichnung der Anweisungsblöcke Querverweise zu allen definierten Namen. Der Benutzer kann einer solchen Strukturliste zwei wichtige Informationen entnehmen:

- An der Stelle, wo der Name definiert ist, werden alle Stellen angezeigt, an denen der Name wieder verwendet wird.
- An den Stellen, wo ein Name verwendet wird, steht ein Verweis auf die Stelle, wo der Name definiert wurde.

Die Strukturliste dient in erster Linie dem Programmierer als Grundlage für die Arbeit am Schreibtisch. Dabei ist die Strukturliste mit Querverweisangaben ein besonders hilfreiches Mittel bei der Erstellung, Änderung und Dokumentation des Quellprogramms. Man erkennt sofort die "Fernwirkungen", Namenskonflikte etc. und vermeidet damit unbeabsichtigte Nebeneffekte bei Änderungen des Quellprogramms. Die Strukturliste mit Querverweisen ist deshalb ein unerläßlicher Bestandteil der Programmdokumentation.

Kennzeichnung der Zeilen

Der Dateiname des eingegebenen Quellprogramms steht in der Kopfzeile der Liste, der Name eines COPY-Elementes ist aus der letzten davorstehenden COPY-Anweisung ersichtlich.
Um das Auffinden der Zeilen aus COPY-Elementen zu erleichtern, ist der Strukturliste eine Liste angehängt, aus der das Kennzeichen und die Zeilennummer jedes COPY-Elements hervorgeht.

Jede Zeile des Programmtextes ist eindeutig gekennzeichnet. Dies erfolgt durch eine fortlaufende Nummer, die sich auf die entsprechende Zeile des eingegebenen Quellprogramms oder COPY-Elements bezieht.

Die Zeilen aus COPY-Elementen sind durch die Buchstaben A - Z und die Ziffern 1 - 9 gekennzeichnet. Kommen mehr als 35 COPY-Elemente vor, werden sie durch eine Buchstaben/Ziffer-Kombination (z.B. "A1" oder "QZ") eindeutig gekennzeichnet.

Verweis von der Definition auf die Verwendung

An der Stelle in der Ausgabeliste, an der ein Name definiert wird, werden in der rechten Spalte alle Zeilen angegeben, an denen dieser Name verwendet wird .

Zusätzlich wird mit folgenden Abkürzungen die Verwendungsart gekennzeichnet:

- bei Datenobjekten:

Leerzeichen	keine Veränderung des Inhalts
*:	Veränderung des Inhalts
F-	Formalparameter in USING-Leiste bei PROCEDURE DIVISION oder ENTRY
R-	Aktualparameter in USING-Leiste bei CALL by reference (Standard)
C-	Aktualparameter in USING-Leiste bei CALL by content
=:	Verwendung in REDEFINES- oder RENAMES-Klausel

- bei Dateien:

I:	OPEN INPUT
O:	OPEN OUTPUT
B:	OPEN I-O
X:	OPEN EXTENDED

- bei Prozeduren (Kapiteln und Paragraphen):

P-	Aufruf mit PERFORM
E-	CALL mit Literal
N-	CALL mit Bezeichner
G-	Ansprung mit GO TO
A-	Verwendung in ALTER
S-	Verwendung in SORT

Innerhalb einer Zeile werden nur die Referenzen eines Typs ausgegeben. Die Referenzen eines Typs werden in der Reihenfolge ausgegeben, in der sie im Programm verwendet werden, also in aufsteigender Nummernfolge innerhalb des Quellprogramms und jedes COPY-Aufrufs.
Sind mehrere Daten innerhalb einer Eingabezeile definiert, so werden Verwendungen zu verschiedenen Daten in verschiedenen Zeilen ausgegeben.

Verweis von der Verwendung auf die Definition

An jeder Stelle, an der ein referenzierter Name verwendet wird, ist die Nummer der Zeile angegeben, in der der Name definiert wurde.
Bei mehreren Verwendungen in einer Zeile werden die Referenzen auf den Ort ihrer Definition in derselben Reihenfolge wie die Verwendungen innerhalb der Zeile ausgegeben.
Bei der Verwendung eines Namens mit Kennzeichnung (z.B. C OF B OF A) wird nur die Definition des hierarchisch untersten Namens (im Beispiel: C) mit der entsprechenden Zeilennummer angegeben.
Ein Fragezeichen ("?") als Referenz kennzeichnet die Verwendung eines nicht definierten Namens.
Ein doppeltes Fragezeichen ("??") bezeichnet einen mehrdeutigen Namen, d.h. es gibt mehrere Definitionen, auf die sich der Name bezieht.

Beispiel 5-3: Strukturliste mit Querverweisangaben
(erzeugt mit INFORMATION-Parameter SEQUENCE-AREA=NO, IDENTIFICATION-AREA=NO sowie mit LAYOUT-Parameter LINE-SIZE=80)

```
  COBOL85 V02.1B00  PRETTY-PRINTING  DATE: 1994-03-03 TIME: 11:36:59 PAGE: 1
  LINE ID   SOURCE: SCHECK.COB                                  REFERENCES

    1        IDENTIFICATION DIVISION.
    2        PROGRAM-ID.
    2            SCHECK.
    3        ENVIRONMENT DIVISION.
    4        INPUT-OUTPUT SECTION.
    5        FILE-CONTROL.
    6            SELECT SCHECKS ASSIGN TO PRINTER.          |D: A-  2
    7            SELECT KONTOS  ASSIGN TO PRINTER.          |D: B-  2
    8        DATA DIVISION.
    9        FILE SECTION.
   10            COPY  SCHECKS.
A   1
A   2        FD  SCHECKS LABEL RECORD STANDARD.              |       6    72
A   2                                                       |   E-  3
A   2                                                      |I:    35
A   3        01  S-DATEN.                                  |   G-  5 I-  5
A   3                                                       |   J-  5
A   4            05  S-NR         PIC 9(7).
A   5            05  S-KONTO-NR   PIC 9(8).               |   D-  4 D-  6
A   5                                                       |   F-  4
A   5                                                      |=: A-  6
A   6            05  FILLER     REDEFINES S-KONTO-NR.      |D: A-  5
A   7                10  STELLE     OCCURS 8 PIC 9.        |   D- 13 D- 17
A   8            05  BETRAG     PIC 9(7)V9(2).            |      46    48
A   8                                                      |      50 C- 12
A   8                                                      |   C- 16 C- 21
A   8                                                      |   C- 54 C- 58
A   8                                                      |   C- 63
A   9
   11            COPY  KONTOS.
B   1
B   2        FD  KONTOS  LABEL RECORD STANDARD.             |       7    72
B   2                                                       |   F-  3
B   2                                                      |I:    35
B   3        01  K-DATEN.                                  |   G-  6
B   4            05  K-ART        PIC XX.                  |   F-  7
B   5            05  K-ZUSTAND    PIC XX.                  |   F-  6
B   6            05  K-KONTO-NR   PIC 9(8).               |   F-  4
B   7            05  GEHALT     PIC 9(7)V9(2).            |   C- 22 C- 64
B   8            05  KREDIT     PIC 9(7)V9(2).            |   C- 14 C- 17
B   8                                                      |   C- 56 C- 59
B   9            05  KONTOSTAND PIC 9(7)V9(2).            |   C- 12 C- 16
B   9                                                      |   C- 22 C- 54
B   9                                                      |   C- 58 C- 64
B  10
   12        WORKING-STORAGE SECTION.
   13        01  CHECK               PIC 9(16).            |   D- 13
   13                                                     |*: D- 11 D- 13
   13                                                     |=:    14
   14        01  FILLER       REDEFINES CHECK.            |D:    13
```

```
 15            05  PRUEFZIFFER  OCCURS 16 PIC 9.                 |   D- 17
 16        77  KONTO-ART            PIC XX.                     |*: F-  7
 17            88  SPEZIAL-CODE        VALUE "SC".               |   C-  6 C- 48
 18            88  GEHALTSKONTO        VALUE "GK".               |   C- 19 C- 61
 19        77  KONTO-ZUSTAND        PIC XX.                     |   K-  5
 19                                                              |*:    42 F-  3
 19                                                              |*: F-  6 F-  9
 20            88  KONTO-GEFUNDEN      VALUE SPACES.
 21            88  KONTO-FEHLT         VALUE "NV".               |       61
 22            88  KONTO-GESPERRT      VALUE "KS".               |   C- 10 C- 52
 23        77  SCHECK-ZUSTAND       PIC XX.                     |   K-  6
 23                                                              |*:    43 D-  8

COBOL85 V02.1B00  PRETTY-PRINTING  DATE: 1994-03-03 TIME: 11:36:59 PAGE: 2
LINE ID   SOURCE: SCHECK.COB                                      REFERENCES

 23                                                              |*: D- 19 D- 21
 23                                                              |*: E-  3
 24            88  KONTO-NR-FALSCH     VALUE "KF".
 25            88  SCHECKS-ZU-ENDE     VALUE "EF".               |       36    39
 26            88  KONTO-NR-OK         VALUE "KO".               |       58 K-  9
 27        77  BETRAG-ZUSTAND       PIC XX.                     |   K-  7
 27                                                              |*:    44    47
 27                                                              |*:    49    51
 27                                                              |*:    53
 28            88  BETRAG-NICHT-NUM    VALUE "BN".
 29            88  BETRAG-OK           VALUE "BO".               |       56 K-  9
 30            88  BETRAG-GESPERRT     VALUE "BS".               |   C-  8 C- 50
 31        77  I                    PIC 9.                      |   D- 12 D- 13
 31                                                              |*: D- 10 D- 14

COBOL85 V02.1B00  PRETTY-PRINTING  DATE: 1994-03-03 TIME: 11:36:59 PAGE: 3
LINE ID   SOURCE: SCHECK.COB                                      REFERENCES

 32        PROCEDURE DIVISION.
 33        STEUERUNG SECTION.
 34        STEUERUNG-1001.
 35            OPEN INPUT SCHECKS KONTOS                         |D: A-  2 B-  2
 36          ┌─PERFORM UNTIL SCHECKS-ZU-ENDE                     |D:    25
 36          ├──────────────────────────────────────────────────
 37          │   PERFORM SCHECK-LESEN                            |D: E-  1
 38          │ ┌─IF
 39          │ │    SCHECKS-ZU-ENDE                              |D:    25
 39          │ ├─THEN─────────────────────────────────────────
 40          │ │   EXIT PERFORM
 41          │ └─END-IF─────────────────────────────────────────
 42          │   MOVE SPACES TO KONTO-ZUSTAND                    |D:    19
 43          │     SCHECK-ZUSTAND                                |D:    23
 44          │     BETRAG-ZUSTAND                                |D:    27
 45          │ ┌─EVALUATE TRUE
 46          │ ├─WHEN BETRAG NOT NUMERIC──────────────────────── |D: A-  8
 47          │ │   MOVE "BN" TO BETRAG-ZUSTAND                   |D:    27
 48          │ ├─WHEN BETRAG < 5──────────────────────────────── |D: A-  8
 49          │ │   MOVE "BS" TO BETRAG-ZUSTAND                   |D:    27
 50          │ ├─WHEN BETRAG > 1000000────────────────────────── |D: A-  8
 51          │ │   MOVE "BS" TO BETRAG-ZUSTAND                   |D:    27
 52          │ ├─WHEN OTHER─────────────────────────────────────
 53          │ │   MOVE "BO" TO BETRAG-ZUSTAND                   |D:    27
 54          │ └─END-EVALUATE─────────────────────────────────────
 55          │   PERFORM KONTO-NR-PRUEFEN                        |D: D-  1
```

```
 56          |  ┌─IF BETRAG-OK                                  |D:    29
 57          |  |    AND
 58          |  |    KONTO-NR-OK                                |D:    26
 59          |  ├─THEN──────────────────────────────────────────
 60          |  |   PERFORM KONTO-LESEN                         |D: F-  1
 61          |  |  ┌─IF KONTO-FEHLT                             |D:    21
 62          |  |  -THEN──────────────────────────────────────
 63          |  |  |   PERFORM FEHLER-MELDUNG                   |D: K-  1
 64          |  |  |   PERFORM SCHECK-ABWEISEN                  |D: I-  1
 65          |  |  ├─ELSE──────────────────────────────────────
 66          |  |  |   PERFORM SCHECK-PRUEFUNG                  |D: C-  1
 67          |  |  └─END-IF──────────────────────────────────────
 68          |  | -ELSE──────────────────────────────────────
 69          |  |   PERFORM FEHLER-MELDUNG                      |D: K-  1
 70          |  └─END-IF──────────────────────────────────────
 71          └─END-PERFORM──────────────────────────────────────
 72            CLOSE SCHECKS KONTOS.                           |D: A-  2 B-  2
 73        STEUERUNG-1002.
 74        <──STOP RUN.
 .
 .
 .
```

```
  COBOL85 V02.1B00  PRETTY-PRINTING  DATE: 1994-03-03 TIME: 11:36:59 PAGE: 10
  LINE ID   SOURCE: SCHECK.COB                                    REFERENCES

TABLE OF ALL INCLUDED COPY ELEMENTS IN THE PROGRAM

          COPY  CALL  IN  COPY ELEMENT
          ID    ID  LINE

          A :         10  SCHECKS
          B :         11  KONTOS
          C :         76  SCHECK-PRUEFUNG
          D :         78  KONTO-NR-PRUEFEN
          E :         80  SCHECK-LESEN
          F :         82  KONTO-LESEN
          G :         84  SCHECK-BEARBEITEN
          H :         86  FEHLER-BEARBEITEN
          I :   H-     1  SCHECK-ABWEISEN
          J :   H-     2  SCHECK-SPERREN
          K :   H-     3  FEHLER-MELDUNG
```

5.3 SDF-Optionen zur Steuerung des COBOL85-Strukturierers

Syntax und Anwendung der SDF-Schnittstelle sind in Kapitel 3 kurzgefaßt beschrieben.

5.3.1 SOURCE-Option

Die Parameter dieser Option bestimmen, ob das Quellprogramm von SYSDTA, aus einer katalogisierten Datei oder aus einer PLAM-Bibliothek eingelesen wird.

Format

```
SOURCE = *SYSDTA / <filename 1..54> / *LIBRARY-ELEMENT(...)
  *LIBRARY-ELEMENT(...)
    | LIBRARY = <filename 1..54 without gen>
    |,ELEMENT = <filename 1..40 without gen-vers>(...)
    |    <filename 1..40 without gen-vers>(...)
    |      | VERSION = *HIGHEST-EXISTING / *UPPER-LIMIT /<alphanum-name 1..24>
```

SOURCE = *SYSDTA
Das Quellprogramm wird von der Systemdatei SYSDTA eingelesen, die im Dialogbetrieb standardmäßig der Datensichtstation zugewiesen ist. Wurde die Quellprogrammdatei vor Beginn des Übersetzunglaufs mit dem ASSIGN-SYSDTA-Kommando der Systemdatei SYSDTA zugewiesen, erübrigt sich die Angabe der SOURCE-Option.

SOURCE = <filename 1..54>
Mit <filename> wird eine katalogisierte Datei zugewiesen.

SOURCE = *LIBRARY-ELEMENT(...)
Mit diesem Parameter wird eine PLAM-Bibliothek und ein Element daraus angegeben.

LIBRARY = <filename 1..54)
Name der PLAM-Bibliothek, in der das Quellprogramm als Element steht.

ELEMENT = <filename 1..40>
Name des Bibliothekselements, in dem das Quellprogramm steht.

VERSION =
Versionsbezeichnung des Bibliothekselements.

VERSION = *HIGHEST-EXISTING
Wird keine Version oder *HIGHEST-EXISTING angegeben, liest der Strukturierer aus dem Element mit der höchsten vorhandenen Versionsnummer.

VERSION = *UPPER-LIMIT
Der Strukturierer liest aus dem Element mit der höchstmöglichen Versionsnummer ("@").

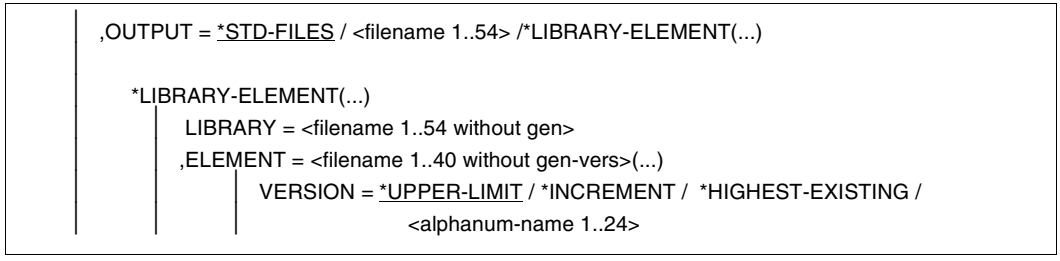
VERSION = <alphanum-name 1..24>
Der Strukturierer liest aus dem Element mit der angegebenen Versionsbezeichnung.

5.3.2 STRUCTURIZER-ACTION-Option

Diese Option bestimmt Informationsumfang, Layout und Ausgabemedium des verwendeten Tools.

Format

```
STRUCTURIZER-ACTION = PRETTY-PRINT(...) / BEAUTIFY(...)

    PRETTY-PRINT(...)
        CROSS-REFERENCE = YES / NO
        ,INFORMATION = STD / PARAMETERS(...)
            PARAMETERS(...)
                SEQUENCE-AREA = YES / NO
                ,IDENTIFICATION-AREA = YES / NO
                ,COPY-EXPANSION = YES / NO
                ,SUB-SCHEMA = YES / NO
                ,LINE-NUMBERS = YES / NO
                ,STATEMENTS = YES / NO

        ,LAYOUT = STD / PARAMETERS(...)
            PARAMETERS(...)
                LOWER-CASE-KEYWORDS = YES / NO
                ,INDENTATION-AMOUNT = 2 / <integer 1..8>
                ,BOLD-FACE = YES / NO
                ,BLOCK-FRAMES = EBCDIC-CHARS / GRAPHIC / NONE
                ,LINES-PER-PAGE = 64 / <integer 20..144> / AS-NEEDED
                ,LINE-SIZE = 132 / <integer 52..132>

        ,OUTPUT = *STD-FILES / <filename 1..54> / SYSLST


    BEAUTIFY(...)
        INFORMATION = STD / PARAMETERS(...)
            PARAMETERS(...)
                SEQUENCE-AREA = YES / NO
                ,IDENTIFICATION-AREA = YES / NO

        ,LAYOUT = STD / PARAMETERS(...)
            PARAMETERS(...)
                LOWER-CASE-KEYWORDS = YES / NO
                ,INDENTATION-AMOUNT = 2 / <integer 1..8>
                ,DATA-FORMATTING = YES/NO
```



STRUCTURIZER-ACTION = PRETTY-PRINT(...)
Es soll eine Strukturliste vom aufbereiteten Quellprogramm erstellt werden.

CROSS-REFERENCE = YES / NO
Bei Angabe von YES wird die Strukturliste mit Querverweisen ergänzt.

INFORMATION = STD
Es werden die voreingestellten Werte der folgenden PARAMETERS-Struktur übernommen.

INFORMATION = PARAMETERS(...)

SEQUENCE-AREA = YES / NO
Bei Angabe von YES wird die "Sequence Number Area" (Spalten 1-6 des Referenzformats) des eingegebenen Quellprogramms übernommen.

IDENTIFICATION-AREA = YES / NO
Bei Angabe von YES wird die "Program Identification Area" (Spalten 73-80 des Referenzformats) des eingegebenen Quellprogramms übernommen.

COPY-EXPANSION = YES / NO
Bei Angabe von YES wird der Inhalt der COPY-Elemente und der SUB-SCHEMA-Section in die Ausgabe übernommen. Das Einlesen der COPY-Elemente wird von der COPY-STATEMENTS-Option gesteuert.

SUB-SCHEMA = YES / NO
Bei Angabe von YES wird der Inhalt der SUB-SCHEMA-Section eingelesen. Die Ausgabe wird vom COPY-EXPANSION-Operanden gesteuert.

LINE-NUMBERS = YES / NO
Bei Angabe von YES wird eine fortlaufende Zeilennumerierung analog zur Quellprogrammliste des COBOL85-Compilers ausgegeben.

STATEMENTS = YES / NO
Bei Angabe von NO werden die Anweisungen nicht ausgegeben. Die Liste enthält dann nur die strukturierten Anweisungen von COBOL85, die Bedingungen sowie die Kommentare. Mit dieser Komprimierung kann man z.B. Implementierungsdetails unterdrücken, so daß die Strukturliste nur den Steuerfluß und die Inline-Dokumentation des Programms zeigt.

LAYOUT = STD
Es werden die voreingestellten Werte der folgenden PARAMETERS-Struktur übernommen.

LAYOUT = PARAMETERS(...)

LOWER-CASE-KEYWORDS = YES / NO
Bei YES werden die vom Strukturierer ergänzten Schlüsselwörter (z.B. explizite Bereichbegrenzer) klein geschrieben. Diese Funktion ist nur sinnvoll, wenn das eingegebene Quellprogramm in Kleinbuchstaben geschrieben ist.

INDENTATION-AMOUNT = 2 / <integer 1..8>
Mit <integer> wird angegeben, um wieviel die Anweisungen innerhalb von Strukturblöcken eingerückt werden sollen.

BOLD-FACE = YES / NO
Bei YES werden die Anweisungen, die einen Strukturblock einleiten, die strukturierenden Zusätze (z.B. THEN) sowie die expliziten Bereichsbegrenzer fett gedruckt.

BLOCK-FRAMES = EBCDIC-CHARS
Die Strukturblockrahmen werden mit folgenden EBCDIC-Zeichen dargestellt:

"|" = X'4F'

"–" = X'60'

"+" = X'4E'

BLOCK-FRAMES = GRAPHIC
Die Strukturblockrahmen werden mit Grafik-Zeichen dargestellt. Der für den Druck verwendete Zeichensatz muß außer den Zeichen für COBOL folgende Zeichen enthalten:

X'22'	für die linke obere Ecke
X'41'	für den senkrechten Strich
X'2E'	für das Mittelstück
X'28'	für die linke untere Ecke
X'3D'	für den waagrechten Strich
X'3A'	für sich kreuzende Striche

BLOCK-FRAMES = NONE
Die Strukturblockrahmen werden nicht dargestellt, d.h. durch Leerzeichen ersetzt.

LINES-PER-PAGE = 64 / <integer 20..144> / AS-NEEDED
Der angegebene Wert bestimmt die Anzahl der Zeilen pro Seite. Bei Angabe von AS-NEEDED wird der Seitenvorschub nicht durch einen Zeilenzähler, sondern nur durch den Beginn von Kapitel (SECTION) oder das Zeichen "/" in Spalte 7 ausgelöst.

LINE-SIZE = 132 / 92 <integer 52..132>
Dieser Wert bestimmt die Anzahl der Zeichen pro Zeile. Voreingestellt ist für Listen mit Querverweisen 132, ohne Querverweise 92. Bei Strukturlisten ohne Querverweise ist die Listenbreite auf 92 Zeichen begrenzt.

OUTPUT = *STD-FILES
Die Liste wird in eine katalogisierte SAM-Datei mit dem Standardnamen STRLST.COB85.program-id-name ausgegeben.

OUTPUT = <filename 1..54>
Das Ergebnis der PRETTY-PRINT-Aktion wird in eine katalogisierte Datei mit dem angegebenen Namen geschrieben.

OUTPUT = *SYSLST
Die Liste wird auf SYSLST ausgegeben.

STRUCTURIZER-ACTION = BEAUTIFY(...)
Das Quellprogramm soll strukturgerecht eingerückt werden.

INFORMATION = STD
Es werden die voreingestellten Werte der folgenden PARAMETERS-Struktur übernommen.

INFORMATION = PARAMETERS(...)

SEQUENCE-AREA = YES / NO
Bei Angabe von YES wird die "Sequence Number Area" (Spalten 1-6 des Referenzformats) in die Ausgabe übertragen.

IDENTIFICATION-AREA = YES / NO
Bei Angabe von YES wird die "Program Identification Area" (Spalten 73-80 des Referenzformats) in die Ausgabe übertragen.

LAYOUT = STD
Es werden die voreingestellten Werte der folgenden PARAMETERS-Struktur übernommen.

LAYOUT = PARAMETERS(...)

LOWER-CASE-KEYWORDS = YES / NO
Bei YES werden die vom Strukturierer ergänzten Schlüsselwörter (z.B. explizite Bereichbegrenzer) klein geschrieben. Diese Funktion ist nur sinnvoll, wenn das eingegebene Quellprogramm in Kleinbuchstaben geschrieben ist.

INDENTATION-AMOUNT = 2 / <integer 1..8>
Mit <integer> wird angegeben, um wieviel die Anweisungen innerhalb von Strukturblöcken eingerückt werden sollen.

DATA-FORMATTING = YES / NO
Bei Angabe von YES werden Dateneinträge in der DATA DIVISION entsprechend der Beschreibung in Abschnitt 5.1 formatiert.
Bei der Angabe von NO wird die Ausgabe der DATA DIVISION nicht formatiert. Dies ist als Unterstützung für jene Programmierer gedacht, die ihre eigenen Formatierungsregeln für die DATA DIVISION weiterverwenden wollen.

OUTPUT = *STD-FILES
Das Ergebnis der BEAUTIFY-Aktion wird in eine katalogisierte Datei geschrieben, deren Name aus dem Namen der Quellprogrammdatei bzw. dem des Bibliothekselements und dem Suffix ".IND" gebildet wird.

OUTPUT = <filename 1..54>
Das Ergebnis der BEAUTIFY-Aktion wird in eine katalogisierte Datei mit dem angegebenen Namen geschrieben.

OUTPUT = *LIBRARY-ELEMENT(...)
Das Ergebnis der BEAUTIFY-Aktion wird in ein Element einer PLAM-Bibliothek geschrieben.

LIBRARY = <filename 1..54>
Name der PLAM-Bibliothek

ELEMENT = <filename 1..40>(...)
Name des Elements in der PLAM-Bibliothek.

VERSION =
Angabe der Versionsbezeichnung

VERSION = *UPPER-LIMIT
Wird keine Versionsbezeichnung oder *UPPER-LIMIT angegeben, erhält das Element die höchstmögliche Versionsnummer ("@").

VERSION = *INCREMENT
Das Element erhält die gegenüber der höchsten vorhandenen Version um 1 inkrementierte Versionsnummer, vorausgesetzt, die höchste vorhandene Versionsbezeichnung endet mit einer Ziffer. Andernfalls ist die Versionsbezeichnung nicht inkrementierbar. In diesem Fall wird *UPPER-LIMIT angenommen.

Beispiel:

höchste vorhandene Version	durch *INCREMENT erzeugte Version
ABC1	ABC2
ABC	@
ABC9	@
ABC09	ABC10
003	004
keine	001

VERSION = *HIGHEST-EXISTING
Die höchste in der Bibliothek vorhandene Version wird überschrieben.

VERSION = <alphanum-name 1..24>
Das Element erhält die angegebene Versionsbezeichnung. Soll die Versionsbezeichnung inkrementierbar sein, muß mindestens das letzte Zeichen eine Ziffer sein (siehe obiges Beispiel).

5.3.3 COPY-STATEMENTS-Option

Mit dieser Option wird die Behandlung von COPY-Anweisungen festgelegt.

Format

```
COPY-STATEMENTS = FLAGGED / IGNORED
```

COPY-STATEMENTS = FLAGGED
Falls ein in einer COPY-Anweisung genanntes Element in keiner der zugewiesenen COPY-Bibliotheken gefunden werden kann, erscheint eine Fehlermeldung.
Die Ausgabe der COPY-Elemente in die Strukturliste steuert der INFORMATION-Parameter COPY-EXPANSION der "Pretty-Print"-Funktion. Bei der "Beautify"-Funktion werden die COPY-Elemente generell nicht ausgegeben.

COPY-STATEMENTS = IGNORED
Es werden nur die COPY-Anweisungen selbst ausgegeben, aber keine COPY-Bibliotheken eröffnet und -elemente eingelesen.

5.3.4 DIAGNOSTICS-Option

Diese Option steuert Umfang und Ausgabemedium der Meldungen.

Format

```
DIAGNOSTICS = YES(...) / NO
    YES(...)
        MINIMAL-WEIGHT = FATAL-ERROR / ERROR / WARNING / NOTE
        ,OUTPUT = STD-FILES / SYSLST
        ,SYSOUT = YES / NO
```

DIAGNOSTICS = YES(...) / NO
Bei YES werden Meldungen ausgegeben.

MINIMAL-WEIGHT = FATAL-ERROR / ERROR / WARNING / NOTE
Es wird festgelegt, ab welchem Gewicht die Meldungen ausgegeben werden. Bei Auftreten eines "Fatal Error" wird der Strukturierungslauf sofort abgebrochen und der Abbruchgrund am Bildschirm gemeldet.

OUTPUT = STD-FILES
Die Meldungen werden in eine katalogisierte Datei mit dem Standardnamen STRERR.COB85.program-id-name ausgegeben. Werden alle Meldungen unterdrückt, enthält die Meldungsdatei nur ihren Namen sowie die Summenzeile über Anzahl und Art der Meldungen.
Endet der Strukturierungslauf, bevor program-id-name ermittelt wurde (z.B. bei einem schwerwiegenden Eingabefehler), so wird anstelle von program-id-name die TSN-Nummer als standardmäßiges Suffix verwendet.

OUTPUT = SYSLST
Die Meldungen werden auf SYSLST ausgegeben.

SYSOUT = YES / NO
Bei Angabe von YES werden alle Meldungen auch auf SYSOUT (Bildschirm) ausgegeben.
Bei Angabe von NO wird am Bildschirm nur die Summenzeile ausgegeben.

5.3.5 MONJV-Option

Mit dieser Option kann eine BS2000-Jobvariable zugewiesen werden.

Format

```
MONJV = *NONE / <filename 1..54 without gen-vers>
```

MONJV = <filename 1..54>
Mit <filename> wird eine überwachende Jobvariable zugewiesen, in die der Strukturierer eine Anzeige über mögliche Ablauffehler ausgibt.

6 Binden, Laden, Starten

Im Verlauf der Übersetzung erzeugt COBOL85 Objektmodule oder Bindelademodule (LLMs), die anschließend in einer PLAM-Bibliothek oder in der temporären EAM-Datei der aktuellen Task zur Verfügung stehen.
Das Programm kann jedoch in dieser Form nicht ablaufen, da sein Maschinencode noch nicht vollständig ist: Jedes Modul enthält Verweise auf externe Adressen, d.h. auf weitere Module, die ihn zur Ausführung ergänzen müssen. Der Compiler erzeugt diese **Externverweise** bei der Übersetzung aus einem oder mehreren der folgenden Gründe:

- Das COBOL-Programm enthält Anweisungen, die
  - komplexe Routinen auf Maschinencode-Ebene erfordern (z.B. SEARCH ALL, INSPECT) oder
  - Schnittstellen zu anderen Softwareprodukten oder zum Betriebssystem bilden (z.B. SORT oder Ein-/Ausgabeanweisungen wie READ, WRITE).

  Dies trifft auf alle COBOL-Programme zu, da in diese Kategorie auch die Routinen zur Programminitialisierung und -beendigung fallen. Die Maschinenbefehlsfolgen für diese Anweisungen werden nicht bei der Übersetzung erzeugt; sie liegen bereits als fertige Module in einer Bibliothek vor, dem **Laufzeitsystem**. Der Compiler trägt für jede solche COBOL-Anweisung in das Modul einen Externverweis auf das zugehörige Modul im Laufzeitsystem ein.
- Das COBOL-Programm ruft ein externes Unterprogramm auf.

  CALL-Anweisungen im Format "CALL literal" veranlassen den Compiler, an den entsprechenden Stellen im Modul Externverweise für den Bindelauf zu erzeugen.
  CALL-Anweisungen im Format "CALL bezeichner" bewirken, daß der dynamische Bindelader die entsprechenden Module zum Ablaufzeitpunkt dynamisch nachlädt (siehe Abschnitt 12.1).
- Das COBOL-Programm ist mit COMOPT GENERATE-SHARED-CODE=YES (in SDF: SHAREABLE-CODE=YES) übersetzt.
  Der Compiler erzeugt ein nicht gemeinsam benutzbares Datenmodul und ein gemeinsam benutzbares Codemodul (siehe 6.7). Im Datenmodul existiert ein Externverweis auf das zugehörige Codemodul.

6.1 Aufgaben des Binders

Der Vorgang, in dessen Verlauf diese Externverweise befriedigt, d.h. die zusätzlich benötigten Module mit dem aus der Übersetzung resultierenden Modul zu einer ablauffähigen Einheit verknüpft werden, heißt Binden; das Dienstprogramm, das diese Aufgabe ausführt, wird als **Binder** bezeichnet.

Ein Binder verarbeitet entweder das Ergebnis einer Übersetzung (Objektmodul oder Bindelademodul) oder ein bereits durch einen Bindelauf vorgebundenes Modul, das ein aus mehreren Objektmodulen bestehendes Großmodul oder ein Bindelademodul sein kann. Objektmodule und Großmodule werden unter dem Begriff "Bindemodul" zusammengefaßt. Dieser Begriff wird im folgenden immer dann verwendet, wenn das zu beschreibende Objekt sowohl ein Objektmodul als auch ein Großmodul sein kann.

Damit die beim Binden erzeugte Einheit ablaufen kann, muß ein **Lader** sie in den Speicher bringen, so daß der Rechner zum Code zugreifen und ihn ausführen kann.

Für die Aufgaben des Bindens und Ladens stehen im **Binder-Lader-Starter-System** des BS2000 folgende Funktionseinheiten zur Verfügung:

- Der Statische Binder TSOSLNK (**TSOS LINK**AGE EDITOR)

  bindet ein oder mehrere Objektmodule zu einem Objektprogramm (auch "Lademodul" genannt) und speichert dieses in einer katalogisierten Datei oder als Element vom Typ C in einer PLAM-Bibliothek,

  oder

  bindet mehrere Objektmodule zu einem einzigen vorgebundenen Modul (Großmodul) und speichert diesen als Element vom Typ R in einer PLAM-Bibliothek oder in der temporären EAM-Datei.

  Es empfiehlt sich, anstelle des Binders TSOSLNK den Binder BINDER zu verwenden, da TSOSLNK nicht weiterentwickelt und durch BINDER abgelöst wurde.

- Der **Binder** BINDER

  bindet Module (Objektmodule, Bindelademodule) zu einer logisch und physisch strukturierten ladbaren Einheit zusammen. Diese Einheit bezeichnet man als "Bindelademodul" (**L**ink and **L**oad **M**odule, **LLM**). Der BINDER speichert den von ihm erzeugten LLM als Element vom Typ L in einer PLAM-Bibliothek.

- Der **D**ynamische **B**inde**l**ader DBL

  fügt in einem Arbeitsgang Module (Objektmodule und Bindelademodule, die ggf. durch einen vorhergehenden Bindevorgang mit dem BINDER erzeugt wurden) einer temporär ladbaren Einheit zusammen, lädt diese sofort in den Speicher und startet sie.

COBOL-Programme, die mindestens ein externes Unterprogramm mit "CALL bezeichner" aufrufen, können nur über dieses Verfahren zum Ablauf gebracht werden (siehe Abschnitt 12.1).

- Der Statische Lader ELDE

  lädt ein Programm, das mit dem TSOSLNK gebunden und in einer Datei oder als Element vom Typ C in einer PLAM-Bibliothek gespeichert wurde.

Der COBOL85-Compiler erzeugt bei der Übersetzung Objektmodule oder LLMs.
Die Objektmodule stehen in der temporären EAM-Datei der aktuellen Task oder als Elemente vom Typ R in einer PLAM-Bibliothek.
Die LLMs stehen als Elemente vom Typ L in einer PLAM-Bibliothek.

Folgende Tabelle zeigt, welche Module von den einzelnen Funktionseinheiten des Binder-Lader-Starter-Systems verarbeitet bzw. erzeugt werden.

Modulart	**Systembaustein**			
	BINDER	DBL	TSOSLNK	ELDE
Objektmodul	ja	ja	ja	nein
Bindelademodul (LLM)	ja	ja *)	nein	nein
Vorgebundener Modul (Großmodul)	ja	ja	ja	nein
Objektprogramm (Lademodul)	nein	nein	ja	ja

*) Nur im Betriebsmodus ADVANCED

Der Bindevorgang im POSIX-Subsystem ist in Kapitel 13 erläutert.

Die folgende Graphik gibt einen Überblick über die verschiedenen Möglichkeiten, temporäre und permanente ablauffähige COBOL-Programme im BS2000 zu erzeugen und aufzurufen:

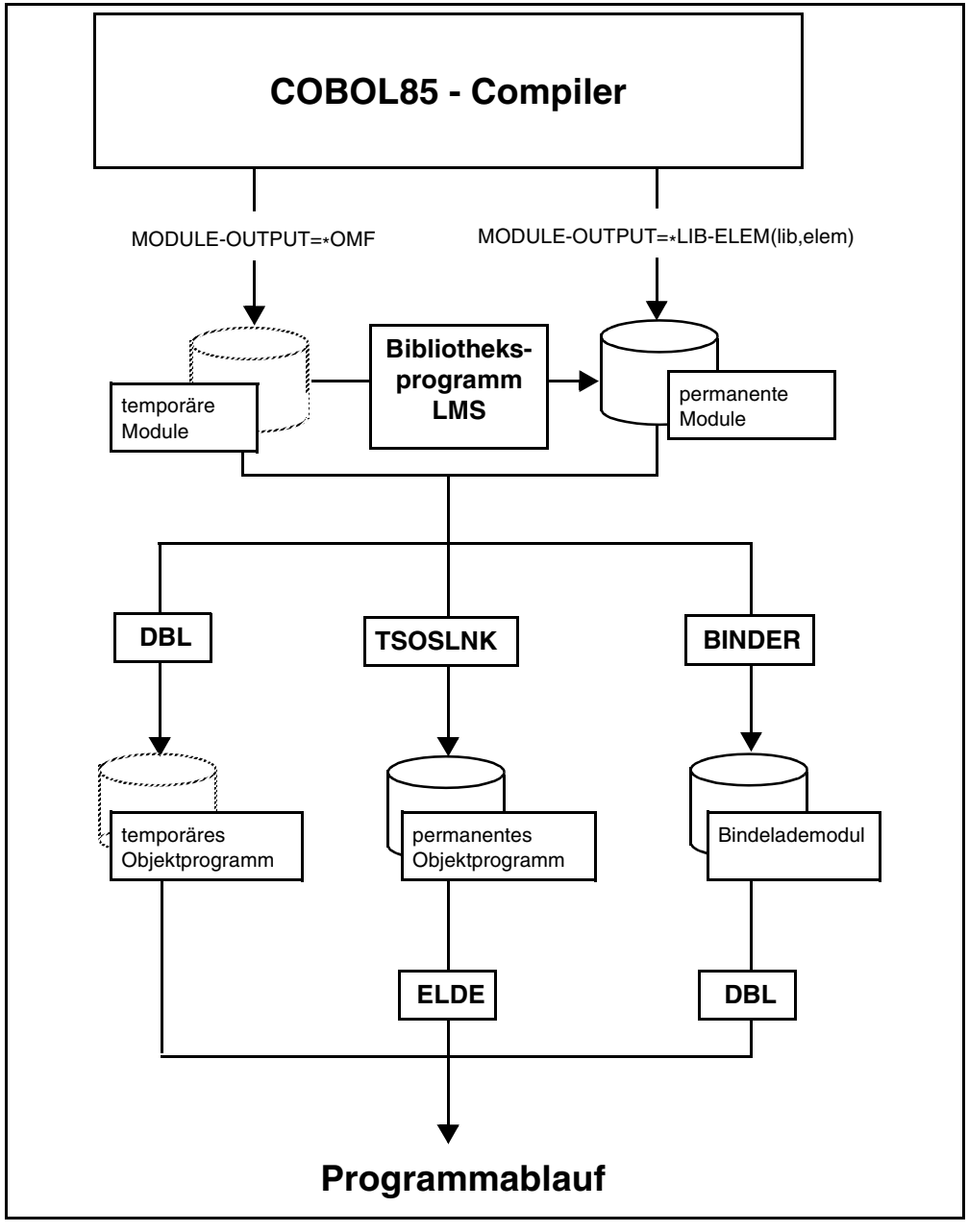


Abbildung 6-1 Erzeugung und Aufruf permanent und temporär ablauffähiger COBOL-Programme im BS2000

6.2 Statisches Binden mit TSOSLNK

Der Statische Binder TSOSLNK erzeugt aus einem oder mehreren Bindemodulen (Objektmodule oder Großmodule) eine der folgenden Einheiten:

– ein ablauffähiges Programm, das er in eine eigene katalogisierte Datei bzw. als Element vom Typ "C" in eine PLAM-Bibliothek ausgibt,

– oder ein vorgebundenes Modul, ein sog. Großmodul, das er in der temporären EAM-Datei der aktuellen Task bzw. als Element vom Typ "R" in einer PLAM-Bibliothek hinterlegt.

Das Dienstprogramm TSOSLNK wird mit dem START-PROGRAM-Kommando aufgerufen. Es erwartet anschließend von SYSDTA Steueranweisungen

- für die Ausgabe, die festlegen,
  - ob das Ergebnis des Binderlaufs ein ablauffähiges Programm oder ein Großmodul sein soll und
  - wohin das Ergebnis ausgegeben werden soll,
- für die Eingabe, die ihm mitteilen,
  - welche Bindemodule er einbinden soll und
  - aus welchen Bibliotheken er offene Externverweise befriedigen soll.

Steueranweisungen für den TSOSLNK

Auf der folgenden Seite finden Sie einen Überblick über die wichtigsten Steueranweisungen des TSOSLNK.

Anweisung	Kurzbeschreibung
PROGRAM PROG	weist den Binder an, aus den eingelesenen Objektmodulen ein Programm zu erzeugen, und legt dessen Eigenschaften und Ausgabeziel (PLAM-Bibliothek oder katalogisierte Datei) fest. Unter anderem können folgende Operanden angegeben werden: – SYMTEST=MAP oder SYMTEST=ALL erlauben es dem Benutzer, beim Testen mit der Dialogtesthilfe AID die symbolischen Namen aus dem Quellprogramm zu verwenden. Voraussetzung dafür ist, daß COBOL85 beim Übersetzen durch eine entsprechende Steueranweisung veranlaßt wurde, LSD-Informationen zu erzeugen. – SYMTEST=ALL weist den Binder an, diese Informationen sofort an das Programm weiterzugeben, während SYMTEST=MAP bewirkt, daß im Testfall LSD-Informationen aus dem Objektmodul nachgeladen werden können (siehe dazu [8]). – LOADPT=*XS legt die Ladeadresse des Programms im Adreßraum oberhalb 16 Mbyte fest. Diese Angabe ist nur möglich, wenn ausschließlich Objektmodule gebunden werden, die in den oberen Adreßraum geladen werden können. – ENTRY/START=einsprungstelle vereinbart den Startpunkt des Programmlaufs. Diese Angabe wird benötigt, falls beim Binden zu einem ablauffähigen Programm das COBOL-Hauptprogramm nicht als erstes eingebunden wird. einsprungstelle ist dann der (ggf. auf 7 Stellen verkürzte) PROGRAM-ID Name mit dem Suffix "$". Die Anweisungen PROGRAM und MODULE (siehe unten) schließen sich gegenseitig aus.
MODULE MOD	veranlaßt den Binder, die eingelesenen Objektmodule zu einem Großmodul zu verknüpfen, und legt dessen Ausgabeziel fest. Die Anweisungen MODULE und PROGRAM (siehe oben) schließen sich gegenseitig aus.
INCLUDE	gibt einzelne Objektmodule an, aus denen der Binder das Programm bzw. das Großmodul aufbauen soll.
RESOLVE	weist TSOSLNK PLAM-Bibliotheken für das (unten beschiebene) Autolink-Verfahren zu.
EXCLUDE	schließt die angegebene PLAM-Bibliothek vom (unten beschriebenen) Autolink-Verfahren aus.
ENTRY	siehe ENTRY- bzw. START-Operand der PROGRAM-Anweisung.
END	markiert das Ende der Eingabe von Binderanweisungen.

Autolink-Verfahren des TSOSLNK

Findet TSOSLNK in einem Modul externe Adreßverweise, die nicht durch die Module befriedigt werden können, die in INCLUDE-Anweisungen angegeben wurden, so geht er nach folgendem **Autolink-Verfahren** vor:

1. Als erstes prüft TSOSLNK, ob dem Externverweis mit einer RESOLVE-Anweisung explizit eine Bibliothek zugeordnet wurde, in der ein passendes Modul zu suchen ist.
2. Kann TSOSLNK im ersten Schritt den Externverweis nicht befriedigen, so durchsucht er sämtliche Bibliotheken, die in RESOLVE-Anweisungen angegeben wurden. Dabei können Bibliotheken durch EXCLUDE-Anweisungen von der Suche ausgeschlossen werden.
3. Ist es TSOSLNK auch im zweiten Schritt nicht gelungen, den Externverweis zu befriedigen, durchsucht er die Bibliothek TASKLIB, sofern dies nicht durch die Anweisung NCAL oder eine entsprechende EXCLUDE-Anweisung verhindert wurde. Falls es unter der Benutzerkennung der aktuellen Task keine Datei namens TASKLIB gibt, verwendet TSOSLNK die Bibliothek des Systems, $.TASKLIB.

Sind auch nach dem Autolink-Verfahren noch unbefriedigte Externverweise vorhanden, gibt TSOSLNK ihre Namen in einer Liste nach SYSOUT und SYSLST aus.

Es ist nicht erlaubt, COBOL-Programme als Klasse-1-Programme zu binden.

Beispiel 6-1: Statisches Binden zu einem ablauffähigen Programm

```
/START-PROGRAM FROM-FILE = $TSOSLNK ------------------------------------------(1)
%  BLS0500 PROGRAM 'TSOSLNK', VERSION 'V21.0E02' OF '1999-03-15' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU TECHNOLOGY SOLUTIONS   2009. ALL
 RIGHTS RESERVED
*PROG COB85PROG,LIB=PLAM.LIB,ELEM=COB85LAD -----------------------------------(2)
*INCLUDE COB85MOD,PLAM.LIB  --------------------------------------------------(3)
*RESOLVE ,$.SYSLNK.CRTE  -----------------------------------------------------(4)
*END  ------------------------------------------------------------------------(5)
%  LNK0500 PROG BOUND
%  LNK0506 PROGRAM LIBRARY : PLAM.LIB
%  LNK0507 ELEMENT WRITTEN : COB85LAD
```

(1) Das Dienstprogramm TSOSLNK wird aufgerufen.

(2) Die PROG-Anweisung legt fest, daß TSOSLNK ein ablauffähiges Programm mit dem Namen COB85PROG erzeugen und als Element unter dem Namen COB85LAD in der PLAM-Bibliothek PLAM.LIB ablegen soll

(3) Die INCLUDE-Anweisung teilt dem Binder mit, daß er das Objektmodul COB85MOD aus der PLAM-Bibliothek PLAM.LIB binden soll.

(4) TSOSLNK soll Externverweise zunächst mit Modulen aus dem Laufzeitsystem befriedigen, das an dieser Anlage unter dem Namen $.SYSLNK.CRTE katalogisiert ist.

(5) END schließt die Eingabe der Steueranweisungen ab und leitet den Bindevorgang ein; nach dessen Abschluß informiert TSOSLNK über das erstellte Programm.

Binden von segmentierten Programmen mit Überlagerungsstruktur

Durch geeignete COBOL-Sprachmittel (siehe [1]) kann der Compiler veranlaßt werden, den Maschinencode für ein Quellprogramm nicht als ein einziges Objektmodul, sondern, in Teile zerlegt, in Form mehrerer Objektmodule auszugeben. Dieser Vorgang heißt **Segmentierung**; die dabei entstehenden Programmteile nennt man **Segmente**.

Beim Binden eines segmentierten Programmes läßt sich eine Überlagerungsstruktur FU-JITSU TECHNOLOGY SOLUTIONS 2009:

Abgesehen vom Root-Segment, das während des gesamten Programmlaufs im Speicher bleibt, kann der Benutzer die einzelnen Segmente programmgesteuert nachladen lassen, wenn sie für den Ablauf erforderlich sind. Dabei können sich Segmente gegenseitig überlagern, d.h. nacheinander einen gemeinsamen Speicherbereich belegen. Welche Segmente einander überlagern können, wird durch Steueranweisungen beim Binden des Programms festgelegt.
Da jedoch der Ablaufteil des BS2000 von sich aus ein Programm in Seiten, d.h. Teile von 4096 Byte, gliedert und bei der Programmausführung jeweils nur die Seiten in den Hauptspeicher lädt, die gerade für den Ablauf benötigt werden, ist im BS2000 Segmentierung zur Entlastung des Hauptspeichers nicht notwendig. Erforderlich wird sie lediglich dann, wenn der virtuelle Adreßraum nicht ausreicht, das gesamte Programm einschließlich der Daten aufzunehmen. Aus diesem Grund ist es nicht möglich, eine echte Überlagerungsstruktur für Programme zu definieren, die auf XS-Anlagen im oberen Adreßraum ablaufen sollen.

Mit folgenden TSOSLNK-Anweisungen lassen sich Überlagerungsstrukturen für segmentierte Programme definieren:

Anweisung	Kurzbeschreibung
OVERLAY	bestimmt die Überlagerungsstruktur für das Programm: Die OVERLAY-Anweisungen eines Binderlaufs legen fest, – welche Segmente einander überlagern können und – an welchen Stellen im Programm sie sich gegenseitig überlagern sollen. OVERLAY-Anweisungen sind nur beim Binden eines Programms erlaubt (PROGRAM-Anweisung); beim Binden eines Großmoduls (MODULE-Anweisung) werden sie mit einer Fehlermeldung zurückgewiesen. Im Adreßraum oberhalb 16 Mbyte (Angabe LOADPT=*XS in der PROGRAM- oder OVERLAY-Anweisung) sind keine echten Überlagerungsstrukturen möglich; der Binder akzeptiert zwar die OVERLAY-Anweisung, ordnet aber die Segmente hintereinander an.
TRAITS	vereinbart für einen Programmteil, daß er – beim Laden auf Seitengrenze ausgerichtet werden soll – während des Programmlaufs nur gelesen werden darf (Angabe READONLY=Y).

6.3 Binden mit dem BINDER

Mit dem BINDER können Objektmodule und Bindelademodule (LLMs) zu einem LLM gebunden und als Element vom Typ L in einer PLAM-Bibliothek abgespeichert werden. Der BINDER ist ausführlich im Handbuch "BINDER" [25] beschrieben.

Beispiel 6-2 Erzeugen eines LLM aus Objektmodulen

```
/START-PROG $BINDER  ------------------------------------------------------------(1)
%  BLS0500 PROGRAM 'BINDER', VERSION 'V02.6A30' OF '2010-10-19' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU TECHNOLOGY SOLUTIONS   2009.
   ALL RIGHTS RESERVED
//START-LLM-CREATION INT-NAME=PROG  ----------------------------------------------(2)
//INCLUDE-MODULES LIB=*OMF,ELEM=MAIN  --------------------------------------------(3)
//INCLUDE-MODULES LIB=PLAM.BSP,ELEM=SUB  -----------------------------------------(4)
//RESOLVE-BY-AUTOLINK LIB=$.SYSLNK.CRTE  -----------------------------------------(5)
//SAVE-LLM LIB=PLAM.BSP,ELEM=TESTPROG  -------------------------------------------(6)
%  BND3101 SOME EXTERNAL REFERENCES UNRESOLVED
%  BND3102 SOME WEAK EXTERNS UNRESOLVED
%  BND1501 LLM FORMAT : '1
//END ----------------------------------------------------------------------------(7)
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED
   EXTERNAL'

/START-PROG *MOD(LIB=PLAM.BSP,ELEM=TESTPROG,RUN-MOD=ADVANCED)  -----------(8)
%  BLS0523 ELEMENT 'TESTPROG', VERSION '@' FROM LIBRARY 'PLAM.BSP' IN
   PROCESSING
%  BLS0524 LLM 'TESTPROG', VERSION ' ' OF '2013-02-26:14:51:46' LOADED
```

(1) Der BINDER wird aufgerufen.

(2) Die Anweisung START-LLM-CREATION erzeugt einen neuen LLM im Arbeitsbereich mit dem internen Namen PROG. Der erzeugte LLM wird später mit der Anweisung SAVE-LLM (siehe 6) als Element vom Typ L in einer PLAM-Bibliothek gespeichert.

(3) Mit dieser INCLUDE-MODULES-Anweisung wird der Name des Moduls angegeben, der das Hauptprogramm enthält (MAIN). Das Modul steht in der temporären EAM-Datei (*OMF).

(4) Mit dieser INCLUDE-MODULES-Anweisung wird der Name des Moduls angegeben, der das Unterprogramm enthält (SUB). Das Modul steht in der PLAM-Bibliothek PLAM.BSP.

(5) Mit der Anweisung RESOLVE-BY-AUTOLINK wird der Name der Laufzeitbibliothek angegeben, aus der Externverweise befriedigt werden sollen.

(6) Mit der Anweisung SAVE-LLM wird der erzeugte LLM unter dem Namen TESTPROG als Element vom Typ L in der PLAM-Bibliothek PLAM.BSP abgespeichert. Die BINDER-Meldung "SOME WEAK EXTERNS UNRESOLVED" bezieht sich auf das ILCS-Modul IT0INITS. Dieses Modul enthält WEAK-EXTERN-Verweise auf alle potentiell für ILCS vorgesehenen Sprachen. Im Beispiel ist nur die Sprache COBOL85 beteiligt, die anderen Verweise bleiben offen.

(7) Mit der END-Anweisung wird der Bindelauf beendet.

(8) Der LLM wird geladen und gestartet.

Bei den Anweisungen INCLUDE-MODULES und RESOLVE-BY-AUTOLINK kann anstelle des Bibliotheksnamens (LIB=bibliothek) auch LIB=*BLS-LINK angegeben werden. In diesem Fall müssen die zu durchsuchenden Bibliotheken mit dem Linknamen BLSLIBnn (00 ≤ nn ≤ 99) zugewiesen werden. Dies geschieht vor Aufruf des BINDERS mit dem SET-FILE-LINK-Kommando, z.B.:

```
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=$.SYSLNK.CRTE
```

Ein mit dem BINDER erzeugter LLM kann - sofern alle Externverweise befriedigt sind - mit dem DBL ohne Zuweisung alternativer Bibliotheken geladen und gestartet werden:

```
START-PROGRAM *MODULE(LIB=bibliothek,ELEM=modul,RUN-MODE=ADVANCED)
```

Achtung !

LLMs mit eingebundenem Laufzeitsystem dürfen nicht in Bibliotheken abgelegt werden, aus denen auch nicht vorgebundene LLMs direkt geladen werden sollen.

Hinweis:

Bei Generierung des LLM-Formats wird eine CSECT mit Namen `programm-name&#` mit folgenden Entries erzeugt:

`programm-name`	für den Unterprogramm-Einsprung
`programm-name&$`	für den Hauptprogramm-Beginn
`programm-name&A`	für den Service-Entry

Bei Generierung von shared-code kommt noch die Code CSECT `progamm-name&@` dazu.

6.4 Dynamisches Binden und Laden mit dem DBL

Mit dem Dynamischen Bindelader DBL werden in einem Arbeitsgang Module temporär zu einer ladbaren Einheit gebunden, dann in den Speicher geladen und gestartet. Die erzeugte Ladeeinheit wird am Ende des Programmablaufs automatisch gelöscht.
Die Arbeitsweise des DBL ist im Handbuch "Bindelader-Starter" [9] ausführlich beschrieben.

Der DBL wird implizit durch die Kommandos START-PROGRAM und LOAD-PROGRAM aufgerufen. Die folgende Übersicht stellt die wichtigsten Angaben der Kommandos START-PROGRAM und LOAD-PROGRAM zum Aufruf des DBL zusammen; die ausführliche Beschreibung aller möglichen Operanden findet sich im Handbuch [9].

```
{START-PROGRAM }                                                  {*OMF,ELEMENT=modul                 }
{              } [FROM-FILE=] *MODULE (LIBRARY= {*OMF [,ELEMENT=*ALL]} )
{LOAD-PROGRAM  }                                {bibliothek,ELEMENT=element}

      [,RUN-MODE = {STD                          } ])
                   {ADVANCED(ALT-LIB=YES)        }
```

Das START-PROGRAM-Kommando weist den Bindelader an, ein ablauffähiges Programm zu erzeugen, es in den Speicher zu laden und zu starten. Da das Programm unmittelbar im Anschluß an das Kommando abläuft, müssen ihm bereits vor dem START-PROGRAM-Kommando die erforderlichen Betriebsmittel (Dateien) zugewiesen werden (siehe Abschnitt 9.1.2).

Das LOAD-PROGRAM-Kommando veranlaßt den Bindelader, ein ablauffähiges Programm zu erzeugen und in den Speicher zu laden, ohne es zu starten. Dadurch lassen sich vor dem Programmablauf weitere Kommandos eingeben - etwa zur Programmüberwachung mit einer Dialogtesthilfe. Das Programm kann daraufhin folgendermaßen gestartet werden:
– durch ein %RESUME-Kommando, falls mit der Dialogtesthilfe AID getestet werden soll oder
– durch ein RESUME-PROGRAM-Kommando in allen anderen Fällen.

LIBRARY=*OMF
bezeichnet die temporäre EAM-Datei der aktuellen Task, in die der Compiler das übersetzte Objektmodul ausgegeben hat.

ELEMENT=modul
gibt den Namen des Moduls an, der zuerst geladen werden soll. modul besteht aus den ersten acht Zeichen des PROGRAM-ID-Namens im Quellprogramm. modul kann auch der Einsprungname (ENTRY-Name) des Programmabschnitts sein, der als erster geladen werden soll.

ELEMENT=*ALL
bewirkt, daß der Bindelader alle Module aus der EAM-Bindemoduldatei holt. Ist dies gewünscht, erübrigt sich die Angabe, da dieser Wert voreingestellt ist.

LIBRARY=bibliothek
gibt den Namen der PLAM-Bibliothek an, in der sich das Modul als Element befindet. Mit *LINK(LINK-NAME=linkname) kann auch ein vereinbarter Linkname für die Bibliothek angegeben werden.

ELEMENT=element
gibt den Namen des Moduls an, der als Element vom Typ R oder L in der angegebenen PLAM-Bibliothek steht. Sind mehrere Elemente gleichen Namens in der Bibliothek gespeichert, wird das Element mit der alphabetisch höchsten Versionsbezeichnung genommen.

RUN-MODE=STD
In diesem Modus muß das Laufzeitsystem CRTE vor Aufruf des Binders mittels SET-TASKLIB-Kommando als TASKLIB zugewiesen werden.
Außer der TASKLIB und ggf. der Bibliothek, die die Module enthält, können keine weiteren Bibliotheken beim Binden berücksichtigt werden.

RUN-MODE=ADVANCED(ALTERNATE-LIBRARIES=YES)
In diesem Modus durchsucht der Binder zur Befriedigung von Externverweisen bis zu 100 verschiedene Bibliotheken, die vor Aufruf des Binders mit dem Linknamen BLSLIBnn ($00 \leq nn \leq 99$) zugewiesen wurden.

Dynamisches Nachladen

Objektmodule mit externen Unterprogrammen, die ausschließlich mit "CALL bezeichner" aufgerufen werden, werden vom DBL zur Ablaufzeit dynamisch nachgeladen. Dazu muß vor Aufruf des DBL die Bibliothek, die die nachzuladenden Objektmodule enthält, mit dem Linknamen COBOBJCT, und das Laufzeitsystem mit einem der Linknamen BLSLIBnn (n = 00 bis 99) zugewiesen werden:

```
/SET-FILE-LINK [LINK-NAME=]COBOBJCT,[FILE-NAME=]bibliothek
/SET-FILE-LINK [LINK-NAME=]BLSLIB00,[FILE-NAME=]$.SYSLNK.CRTE
```

Die Verwendung des Linknamens BLSLIBnn ist nur möglich, wenn im Aufrufkommando RUN-MODE=ADVANCED(ALTERNATE-LIBRARIES=YES) angegeben ist.

Siehe auch Kapitel 12.1, Beispiel 12-1.

Beispiel 6-3: Dynamisches Binden und Laden eines Moduls aus einer PLAM-Bibliothek

```
/START-PROGRAM *MOD(LIB=PLAM.BSP,ELEM=COB85MOD) ------------------------- (1)
%  BLS0001 ### DBL VERSION 070 RUNNING
%  BLS0335 UNRESOLVED EXTERNAL REFERENCES 'ITCN021D ITCSACA0 ITCSBEG0
ITCSDSA0 ITCSEND0'  ---------------------------------------------------- (2)

%  BLS0336 CONTINUE PROCESSING? REPLY (Y=YES; N=NO)?
N
%  NRTT101 ABNORMAL JOBSTEP TERMINATION BLS0532
/SET-TASKLIB $.SYSLNK.CRTE  -------------------------------------------- (3)
/START-PROGRAM *MOD(LIB=PLAM.BSP,ELEM=COB85MOD)
%  BLS0001 ### DBL VERSION 070 RUNNING
%  BLS0517 MODULE 'COB85MOD' LOADED
```

(1) Das START-PROGRAM-Kommando weist den Bindelader an, das Modul COB85MOD aus der PLAM-Bibliothek PLAM.LIB zu binden, zu laden und anschließend zu starten. Zuvor wurde keine TASKLIB vereinbart.

(2) Der Bindelader meldet, daß Externverweise zu COBOL85-Laufzeitmodulen (ITC...) nicht befriedigt werden können; d.h. sie sind weder in einer Bibliothek namens TASKLIB, noch unter der Benutzerkennung der aktuellen Task, noch in der $.TASKLIB vorhanden.

(3) An dieser Anlage ist das Laufzeitsystem unter dem Namen $.SYSLNK.CRTE katalogisiert. Nachdem es mit einem SET-TASKLIB-Kommando zur TASKLIB erklärt worden ist, kann das nachfolgende START-PROGRAM-Kommando erfolgreich ausgeführt werden, und das Programm läuft ab.

6.5 Laden und Starten von ablauffähigen Programmen

Damit ein statisch gebundenes Programm ablaufen kann, muß es in den Hauptspeicher geladen werden. Für diese Aufgabe steht im BS2000 ein Statischer Lader zur Verfügung. Er wird - wie der Dynamische Bindelader mit den Kommandos START-PROGRAM bzw. LOAD-PROGRAM (siehe [9]) aufgerufen:

- Das START-PROGRAM-Kommando weist den Lader an, das Programm in den Speicher zu laden und zu starten. Da das Programm unmittelbar im Anschluß an das Kommando abläuft, müssen ihm bereits vorher die erforderlichen Betriebsmittel (Dateien) zugewiesen werden (siehe Abschnitt 9.1.2).
- Das LOAD-PROGRAM-Kommando weist den Lader an, das Programm in den Speicher zu laden, ohne es zu starten. Dadurch lassen sich vor dem Programmablauf weitere Kommandos eingeben - etwa zur Programmüberwachung mit einer Dialogtesthilfe. Das Programm kann dann mit einem RESUME-PROGRAM- oder %RESUME-Kommando gestartet werden.

Die folgende Übersicht stellt die wichtigsten Angaben der Kommandos START-PROGRAM und LOAD-PROGRAM für den Aufruf des Statischen Laders zusammen; eine ausführliche Beschreibung findet sich im Handbuch "Binder-Lader-Starter" [9].

```
{START-PROG}                 {*PHASE(LIB=bibliothek,ELEM=element,VERS=version)}
{          } FROM-FILE = {                                                    }
{LOAD-PROG }                 {dateiname                                        }
```

---

bibliothek — gibt den Namen einer PLAM-Bibliothek an, die das vom TSOSLNK erzeugte Programm als Element enthält.

element — ist der Name des Bibliothekselements, in dem das Programm gespeichert ist. Das Element muß vom Typ C sein.

version — gibt eine Elementversion mit maximal 24 Zeichen Länge an.

dateiname — ist der Name der katalogisierten Datei, die das vom TSOSLNK erzeugte Programm enthält.

6.6 Programmbeendigung

Das Beendigungsverhalten eines Programms ist insbesondere dann von Bedeutung, wenn es in einer Prozedur aufgerufen oder von einer Jobvariablen überwacht wird.

Treten während des Programmablaufs Fehlermeldungen auf, denen ein interner Return-Code zugeordnet ist (siehe dazu auch Fehlermeldung COB9119 in Kap.15), wird dieser Return-Code in die letzten beiden Bytes der Rückkehrcode-Anzeige einer überwachenden Jobvariablen (siehe [7]) übernommen.

Die folgende Tabelle gibt einen Überblick über

– die möglichen Inhalte der Rückkehrcode-Anzeige in Jobvariablen,

– die zugeordneten Fehlermeldungen und

– deren Auswirkung auf den weiteren Verlauf einer Prozedur.

Rückkehr-Code-Anzeige [1]	**Fehler-nummer** [2]	**Kurzbeschreibung des Fehlers**	**Fortsetzung steuerbar mit Option** [3]	**Dump**	**Verhalten in Prozeduren**
01**00**	keine	Vom Laufzeitsystem wurde kein Fehler erkannt	---	nein	keine Verzweigung
11**20**	COB9120	Jobvariablen nicht verfügbar	ja		Verzweigung zum nächsten STEP-, ABEND-, ABORT- oder LOGOFF-Kommando
11**21** 11**22**	COB9121 COB9122	End of File bei ACCEPT	ja ja		
11**23** 11**24** 11**25** 11**26** 11**27**	COB9123 COB9124 COB9125 COB9126 COB9127	fehlerhaftes Argument in einer Standardfunktion	ja ja ja ja ja		
11**28**	COB9128	Anwender-Returncode (Users Return Code) ist gesetzt	nein		
11**31**	COB9131	Jobvariablen: ACCEPT auf leere Jobvariable	ja		
11**32**	COB9132	falsche Parameteranzahl (CALL)	ja		
11**33**	COB9133	Programmablauf in BS2000-Version < 10,0	nein		
11**34**	COB9134	Sort-Fehler	ja		
21**40**	COB9140	fehlerhafte Teilfeldselektion	ja	nein/ [4] ja	
21**42**	COB9142	GO TO ohne ALTER	nein		
21**43**	COB9143	Freigabedatum für Datenträger noch nicht erreicht	nein		
21**44**	COB9144	Tabelle: Subskript-/Indexbereich überschritten	ja		
21**45**	COB9145	Tabelle (mit DEPENDING ON-Element): Subskript-/Indexbereich überschritten	ja		
21**46**	COB9146	COBOL85-Laufzeitsystem ist inkompatibel zum Objektprogramm	nein		
21**48**	COB9148	CALL nicht ausführbar	nein		
21**49**	COB9149	Inkompatible Daten in numerisch editiertem Feld	nein		

Tabelle 6-1: Rückkehrcode-Anzeige in Jobvariablen

Rückkehr-Code-Anzeige [1]	Fehler-nummer [2]	Kurzbeschreibung des Fehlers	Fortsetzung steuerbar mit Option [3]	Dump	Verhalten in Prozeduren
21**51**	COB9151	Dateien: Nicht abgefangener Ein-/ Ausgabe-fehler (keine USE-Prozedur, kein INVALID KEY, kein AT END)	nein	nein/ [4] ja	Verzweigung zum nächsten STEP-, ABEND-, ABORT- oder LOGOFF-Kommando
21**52**	COB9152	Verbindung zu Datenbank konnte nicht hergestellt werden	nein		
21**54**	COB9154	REPORT WRITER: Anwenderfehler	nein		
21**55**	COB9155	Fehler beim Verlassen einer USE-Prozedur	nein		
21**56**	COB9156	DML: Zu kleines SUB-SCHEMA-Modul zur Verarbeitung einer umfangreichen DML-Anweisung	nein		
21**57**	COB9157	CALL nicht ausführbar	nein		
21**58**	COB9158	Mehr als 9 rekursive Aufrufe von DEPENDING-Paragraphen	nein		
21**60**	COB9160	Ablaufeinheit verwendet CANCEL, enthält aber Programme, die mit einem Compiler < V2.0 übersetzt wurden	nein		
21**62**	COB9162	Die Eigenschaften einer externen Datei sind in den Programmen einer Ablaufeinheit nicht konsistent	nein		
21**63**	COB9163	Der Speicherplatz für DYNAMIC-Daten konnte nicht angelegt werden	nein		
21**64**	COB9164	Mit CALL aufgerufenes Programm ist nicht verfügbar	nein		
21**68** 21**69** 21**71**	COB9168 COB9169 COB9171	REPORT WRITER: Anwenderfehler	nein nein nein		
2173	COB9173	SORTlauf nicht erfolgreich	nein		
21**74** 21**75**	COB9174 COB9175	Fehlerbehandlung im Programm: Anwenderfehler	nein nein		
21**76**	COB9176	REPORT WRITER: Anwenderfehler			

Tabelle 6-1: Rückkehrcode-Anzeige in Jobvariablen

Rückkehr-Code-Anzeige [1]	**Fehler-nummer** [2]	**Kurzbeschreibung des Fehlers**	**Fortsetzung steuerbar mit Option** [3]	**Dump**	**Verhalten in Prozeduren**
21**78**	COB9178	Zu sortierender Satz paßt nicht zu SD-Beschreibung	nein	nein/ [4] ja	Verzweigung zum nächsten STEP-, ABEND-, ABORT- oder LOGOFF-Kommando
21**79**	COB9179	sortierter Satz paßt nicht zur GIVING-Dateibeschreibung	nein		
21**80**	COB9180	RELEASE / RETURN außerhalb der SORT-/ MERGE-Steuerung	nein		
21**81**	COB9181	DATABASE-HANDLER hat letzte DML-Anweisung noch nicht abgearbeitet	nein		
21**84**	COB9184	SORT innerhalb der SORT-Steuerung	nein		
31**92**	COB9192	Programmende wurde erreicht, ohne daß STOP RUN oder EXIT PROGRAM ausgeführt wurde	nein	ja	
31**93**	COB9193	Fehler bei DISPLAY	nein		
31**94**	COB9194	Fehler bei Eingabe von SYSDTA	nein		
31**95**	COB9195	Fehler bei Ausgabe auf SYSLST	nein		
31**96**	COB9196	ACCEPT- oder DISPLAY-Anweisung: Fehler an der Schnittstelle Laufzeitystem-Betriebssystem	nein		
31**97**	COB9197	Jobvariablen: fehlerhafter Zugriff	ja		
31**98**	COB3198	Hardware-Unterbrechung	nein		
31**99**	keine	WROUT-Fehler: Es kann keine Meldung mehr ausgegeben werden	nein		

Tabelle 6-1: Rückkehrcode-Anzeige in Jobvariablen

[1] Die 1. Ziffer bezeichnet das Gewicht der Meldung (0: Hinweis, 1: Warnung, 2: Fehler, 3: Abbruchfehler).
Die 2. Ziffer (immer 1) kennzeichnet das Programm als COBOL-Objekt.
Die beiden letzten Ziffern (fett gedruckt) stellen den internen Return-Code dar.

[2] Inhalt und Bedeutung der Meldungen siehe Kapitel 15

[3] Mit RUNTIME-OPTIONS=PAR(ERROR-REACTION = TERMINATION) bzw. COMOPT CONTINUE-AFTER-MESSAGE=NO kann der Programmabbruch herbeigeführt werden. Nach Programmabbruch wird der dazugehörige Rückkehrcode in die progamm-überwachende Jobvariable gesetzt.

[4] Stapelbetrieb: nein
Dialogbetrieb: Abfrage ja/nein

6.7 Gemeinsam benutzbare COBOL-Programme

Bei großen Programmen kann es von Vorteil sein, einzelne Programmteile, auf die mehrere Benutzer (Tasks) zugreifen, gemeinsam benutzbar (shareable) zu machen.

Hierfür ist bei der Übersetzung eine der folgenden Steueranweisungen anzugeben:

COMOPT GENERATE-SHARED-CODE=YES

oder

SHAREABLE-CODE=YES
im MODULE-GENERATION-Parameter der COMPILER-ACTION-Option.

Der Compiler erzeugt dann zwei Objektmodule, wovon das eine den nicht mehrfachbenutzbaren Teil und das andere den gemeinsam benutzbaren Teil des Objekts enthält. Sie werden im folgenden als "nicht gemeinsam benutzbares" bzw. "mehrfachbenutzbares Modul" bezeichnet. Die gemeinsam benutzbaren bzw. nicht mehrfachbenutzbaren Module können jeweils zu Großmodulen vorgebunden werden.

Die gemeinsam benutzbaren Module müssen entweder unmittelbar vom Compiler (über COMOPT-Anweisung MODULE bzw. SDF-Option MODULE-LIBRARY) oder mit dem Dienstprogramm LMS (siehe [10]) in einer PLAM-Bibliothek abgelegt werden.
Diese gemeinsam benutzbaren Module erklärt der Systemverwalter mit dem ADD-SHARED-PROGRAM-Kommando als "shareable" und lädt sie - für alle Tasks verfügbar - in den Klasse-4-Speicher.
Alle nicht gemeinsam benutzbaren Teile eines Programms werden pro Task und Anwender in den Klasse-6-Speicher geladen.
Programmsysteme mit gemeinsam benutzbaren Modulen können nur mit dem Dynamischen Bindelader aufgerufen werden. Aufgerufen wird stets der Name des nicht gemeinsam benutzbaren (Daten)-Moduls. Dieses enthält Externverweise auf sein gemeinsam benutzbares Codemodul sowie ggf. auf andere nicht gemeinsam benutzbare Module.

Aufrufbeispiel:

```
/SET-TASKLIB $.SYSLNK.CRTE ————————————————————————————————————(1)
/START-PROGRAM *MOD(bibliothek,element) ———————————————————————(2)
```

(1) Mit dem SET-TASKLIB-Kommando wird die Bibliothek zugewiesen, die die COBOL85-Laufzeitmodule enthält.

(2) element ist der Name des Datenmoduls oder Großmoduls, das mindestens den nicht gemeinsam benutzbaren Teil des Hauptprogramms enthalten muß. bibliothek ist die Bibliothek, in der die vom Benutzer geschriebenen Module stehen.

Das folgende Bild veranschaulicht Programmläufe ohne und mit "Shared Code":

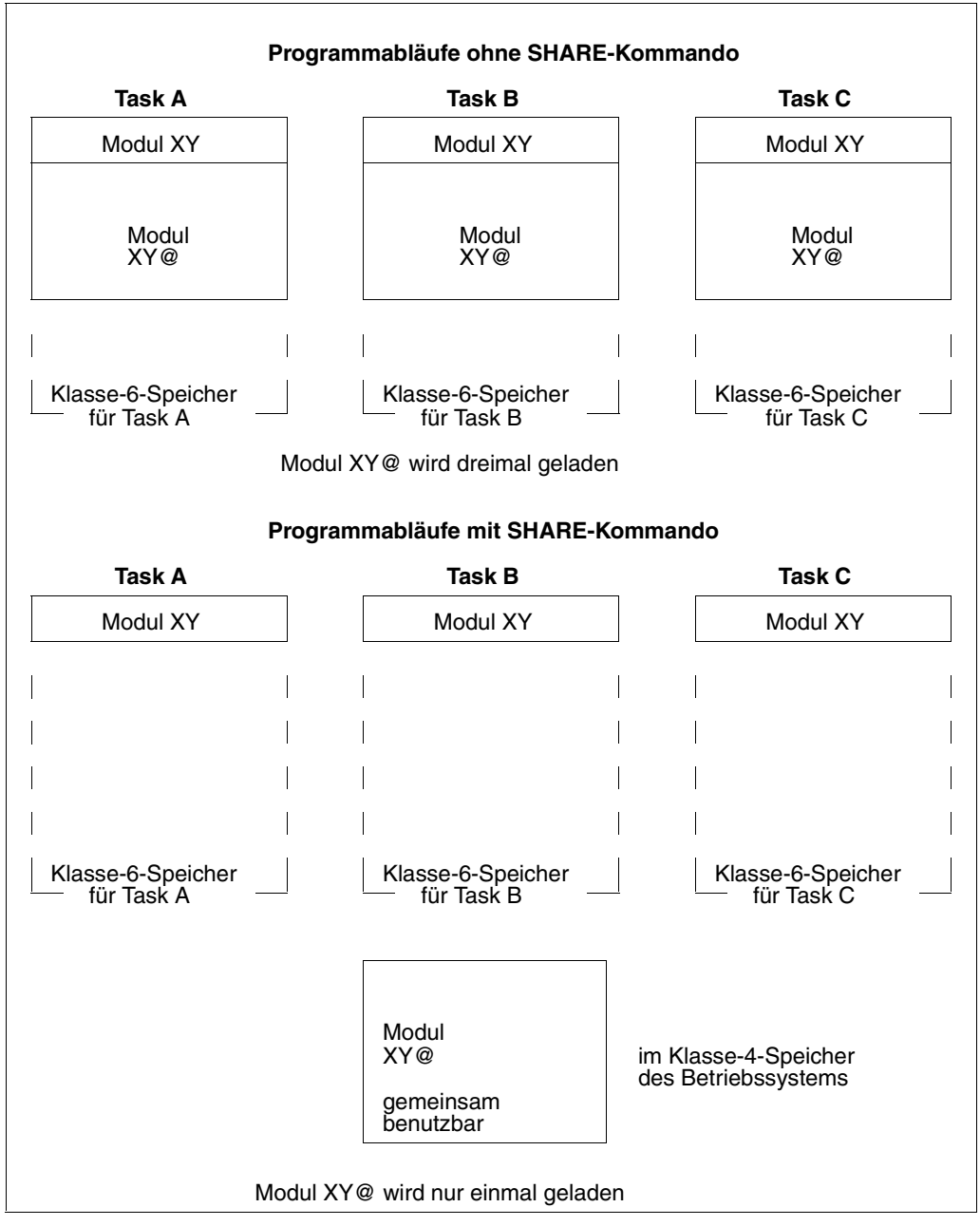


Abbildung 6-2 Shared Code

7 Testhilfen für den Programmablauf

Auch ein syntaktisch korrektes COBOL-Programm enthält möglicherweise noch logische Fehler und läuft daher nicht in der gewünschten Weise ab. Für das Auffinden und Beseitigen solcher Fehler stehen dem COBOL-Programmierer verschiedene Hilfsmittel zur Verfügung:

– Er kann während des Programmlaufes die Dialogtesthilfe AID (**A**dvanced **I**nteractive **D**ebugger) einsetzen. Sie erfordert keine Vorkehrungen bei der Programmierung und erlaubt es, im geladenen Programm während dessen Ausführung Fehler zu suchen und korrigierend in den Ablauf einzugreifen.

– Er kann bereits in das Quellprogramm Testhilfezeilen einbauen und sie bei Bedarf aktivieren. Dies setzt voraus, daß schon bei der Erstellung des Quellprogramms mögliche Fehlersituationen eingeplant werden. Die Diagnose unvorhergesehener Fehler kann es daher erforderlich machen, Testhilfezeilen abzuändern oder hinzuzufügen und anschließend das Quellprogramm neu zu übersetzen. Testhilfezeilen werden in [1] und in Abschnitt 7.2 beschrieben.

Die Testhilfen lassen sich im POSIX-Subsystem analog verwenden (siehe Kapitel 13).

7.1 Dialogtesthilfe AID

In COBOL85-BC nicht unterstützt !

In diesem Benutzerhandbuch soll AID lediglich kurz vorgestellt werden. Die ausführliche Beschreibung dieser Testhilfe findet sich in den Handbüchern [8], [22] und [23].

AID zeichnet sich durch folgende Leistungsmerkmale aus:

1. Es bietet die Möglichkeit, "symbolisch" zu testen, d.h. in den Kommandos anstelle absoluter Adressen auch symbolische Namen aus dem Quellprogramm anzugeben, wenn die dafür nötigen LSD-Informationen beim Übersetzen erzeugt und später an das geladene Programm weitergegeben werden (siehe 7.1.2).

   Dabei ist es nicht unbedingt erforderlich, diese Informationen stets für das Gesamtprogramm zusammen mit diesem Programm zu laden. AID erlaubt nämlich ein Nachladen der LSD-Informationen für jede Übersetzungseinheit, falls die zugehörigen Module (mit den LSD-Informationen) in einer PLAM-Bibliothek stehen. Dadurch lassen sich Betriebsmittel wirtschaftlicher einsetzen:

   – Der Programmspeicher wird entlastet, da LSD-Informationen nur dann geladen werden müssen, wenn sie zum Testen benötigt werden (der Speicherbedarf für ein Programm steigt durch das Mitladen dieser Informationen ungefähr auf das Fünffache).

   – Ein Programm, das im Test fehlerfrei bleibt, muß für den Produktiveinsatz nicht unbedingt neu (ohne LSD-Informationen) übersetzt oder gebunden werden.

   – Falls sich für ein Programm während seines Produktiveinsatzes ein Test als nötig erweist, stehen dafür LSD-Informationen zur Verfügung, ohne daß das Programm erneut übersetzt und gebunden werden muß.

2. Es stellt Funktionen zur Verfügung, die es insbesondere gestatten,

   – den Programmablauf auf symbolischer Ebene zu verfolgen und zu protokollieren (TRACE-Funktion),

   – den Programmablauf an festgelegten Stellen oder beim Eintreten definierter Ereignisse zu unterbrechen, um AID- oder BS2000-Kommandos (sogenannte Subkommandos) ausführen zu lassen,

   – nach einer Programmunterbrechung ein Kapitel oder einen Paragraphen der PROCEDURE DIVISION zu vereinbaren, mit dem - abweichend von der codierten Programmlogik - der Testablauf fortgesetzt werden soll (%JUMP-Anweisung (siehe [8])); nur möglich, wenn das Programm mit PREPARE-FOR-JUMPS=YES im AID-Parameter der TEST-SUPPORT-Option bzw. mit
   COMOPT SEPARATE-TESTPOINTS=YES übersetzt wurde
   (siehe Abschnitte 3.3.7 bzw. 4.2),

– sich die Inhalte von Feldern in einer Form ausgeben zu lassen, welche die Datendefinitionen des Quellprogrammes berücksichtigt,

– die Inhalte von Feldern zu verändern, wobei AID die dazu nötigen Datenübertragungen gemäß den Regeln der COBOL-MOVE-Anweisung durchführt.

3. Es unterstützt neben der Diagnose geladener Programme auch die Analyse von Speicherabzügen in Plattendateien.
4. Es kann im Dialog- und im Stapelbetrieb eingesetzt werden. Für einen Programmtest empfiehlt sich allerdings der Dialog, da die Folge der Kommandos nicht im voraus festgelegt werden muß und der jeweiligen Testsituation angepaßt werden kann.

7.1.1 Voraussetzungen für das symbolische Testen

Beim Testen auf symbolischer Ebene erlaubt es AID, Datenfelder, Kapitel und Paragraphen mit den im Quellprogramm definierten Namen anzusprechen und sich auf Anweisungszeilen und einzelne COBOL-Verben in der PROCEDURE DIVISION zu beziehen. Dafür müssen AID Informationen über diese symbolischen Namen zur Verfügung gestellt werden. Diese Informationen gliedern sich in zwei Teile (siehe dazu [24]),

– die LSD (List for Symbolic Debugging), in der die im Modul definierten symbolischen Namen und Anweisungen verzeichnet sind und

– das ESD (External Symbol Dictionary), das die Externbezüge eines Moduls registriert.

Die Erzeugung bzw. Weitergabe dieser Informationen wird durch entsprechende Operanden im Aufrufkommando bzw. in der Steueranweisung bei jedem der folgenden Schritte veranlaßt oder unterdrückt:

– Übersetzen mit COBOL85

– Binden und Laden mit dem Dynamischen Bindelader oder

– Binden mit dem Statischen Binder und

– Laden mit dem Statischen Lader

Dabei werden ESD-Informationen standardmäßig generiert und weitergegeben, während die LSD-Informationen AID auf zwei Wegen zugänglich gemacht werden können: Nachdem sie bei der Übersetzung erzeugt worden sind, ist es möglich,

– sie zusammen mit dem Gesamtprogramm zu laden oder

– sie erst bei Bedarf für jede Übersetzungseinheit nachzuladen, falls die zugehörigen Module in einer PLAM-Bibliothek stehen.

Die folgende Tabelle gibt für beide Fälle einen Überblick über die Operanden, die zur Erzeugung und Weitergabe der LSD-Informationen angegeben werden müssen.

Schritte in der Programmentwicklung	**Operanden - Angabe**	
	wenn die LSD-Information zusammen mit dem Gesamtprogramm geladen werden soll	wenn später die LSD-Information durch AID nachgeladen werden soll [1]
Übersetzen mit COBOL85	TEST-SUPPORT=AID() oder COMOPT SYMTEST=ALL	TEST-SUPPORT=AID() oder COMOPT SYMTEST=ALL
Binden und Laden mit dem Dynamischen Bindelader	LOAD-PROGRAM ..., TEST-OPTIONS=AID oder START-PROGRAM ..., TEST-OPTIONS=AID	LOAD-PROGRAM ..., [TEST-OPTIONS=NONE] oder START-PROGRAM ..., [TEST-OPTIONS=NONE]
Binden mit TSOSLNK	PROGRAM...,SYMTEST=ALL	PROGRAM...[,SYMTEST=MAP]
Laden bzw. Laden und Starten mit dem Statischen Lader	LOAD-PROGRAM ..., TEST-OPTIONS=AID oder START-PROGRAM ..., TEST-OPTIONS=AID	LOAD-PROGRAM ..., [TEST-OPTIONS=NONE] oder START-PROGRAM ..., [TEST-OPTIONS=NONE]

Tabelle 7-1: Operanden zur Erzeugung von LSD-Informationen

[1] Dies ist nur dann möglich, wenn die zugehörigen Module in einer PLAM-Bibliothek stehen.

Informationen über das Testobjekt

Mit dem AID-Kommando

```
%D[ISPLAY] { _COMPILER          }
           { _COMPILATION_DATE  }
           { _COMPILATION_TIME  }
           { _PROGRAM_NAME      }
```

können allgemeine Informationen über das zu testende Objekt angefordert werden:

`_COMPILER`	Compiler, von dem das Objekt übersetzt wurde
`_COMPILATION_DATE`	Datum der Übersetzung
`_COMPILATION_TIME`	Uhrzeit der Übersetzung
`_PROGRAM_NAME`	PROGRAM-ID-Name des Objekts

7.1.2 Symbolisches Testen mit AID

Beim symbolischen Testen mit AID können Datenfelder, Kapitel und Paragraphen mit den Namen angesprochen werden, die im Quellprogramm definiert wurden.

Um dagegen auf eine beliebige Zeile in der PROCEDURE DIVISION Bezug zu nehmen, muß der Benutzer einen Namen der Form

– S'n' (für eine Zeile mit einem Kapitel- oder Paragraphennamen) bzw.

– S'nverbm' (für eine Zeile mit COBOL-Verben)

angeben. Einen solchen **LSD-Namen** bildet COBOL85 für jede Zeile in der PROCEDURE DIVISION und für jedes COBOL-Verb in einer Anweisungszeile (siehe Beispiel 7-1). Seine Bestandteile haben dabei folgende Bedeutung:

n ist die maximal fünfstellige Nummer dieser Zeile in der PROCEDURE DIVISION, die COBOL85 bei der Übersetzung vergeben hat. Sie muß ohne führende Nullen angegeben werden. Soll als Zeilennummer die (maximal sechsstellige) Folgenummer des Quellprogramms verwendet werden, muß der Benutzer dies mit dem SDF-Operanden STMT-REFERENCE=COLUMN-1-TO-6 in der TEST-SUPPORT-Option bzw. mit COMOPT TEST-WITH-COLUMN1 anfordern.

verb ist die festgelegte Abkürzung eines COBOL-Verbs in der betreffenden Zeile. Diese Abkürzungen können der nachstehenden Liste entnommen werden.

m ist eine einstellige Nummer, die angibt, das wievielte von mehreren gleichen COBOL-Verben innerhalb der Zeile n bezeichnet werden soll.
Falls m gleich 1 ist, wird es weggelassen.

Beispiel 7-1: Bildung von LSD-Namen

```
000026          IF A = B MOVE A TO D MOVE B TO E.
```

In dieser Anweisungszeile hat

– das erste Verb den LSD-Namen S'26IF',

– das zweite Verb den LSD-Namen S'26MOV',

– das dritte Verb den LSD-Namen S'26MOV2'.

Ein ausführliches Beispiel für das Testen eines COBOL-Programms mit AID findet sich im AID-Handbuch "Testen von COBOL-Programmen" [8].

Liste der COBOL-Verben und ihrer Abkürzungen:

ACC	ACCEPT	INI	INITIATE
ADD	ADD	INSP	INSPECT
ADDC	ADD CORRESPONDING	KEE	KEEP
ALT	ALTER	MOD	MODIFY
CALL	CALL	MOV	MOVE
CANC	CANCEL	MOVC	MOVE CORRESPONDING
CLO	CLOSE	MRG	MERGE
COM	COMPUTE	MUL	MULTIPLY
CON	CONNECT	OPE	OPEN
CONT	CONTINUE	PER	PERFORM oder EXIT PERFORM
DEL	DELETE		oder Ende des Schleifenrumpfes [2]
DIS	DISPLAY	PERT	TEST OF PERFORM
DIV	DIVIDE	REA	READ
DSC	DISCONNECT	REDY	READY
END	END-xxx [1] [2]	REL	RELEASE
		RET	RETURN
ERA	ERASE	REW	REWRITE
EVAL	EVALUATE	SEA	SEARCH
EXI	EXIT	SET	SET
EXIT	EXIT PROGRAM	SOR	SORT
FET	FETCH	STA	START
FIN	FINISH	STO	STOP
FND	FIND	STOR	STORE
FRE	FREE	STRG	STRING
GEN	GENERATE	SUB	SUBTRACT
GET	GET	SUBC	SUBTRACT CORRESPONDING
GOT	GO TO	TER	TERMINATE
IF	IF	UNST	UNSTRING
INIT	INITIALIZE	WRI	WRITE

[1] expliziter Bereichsbegrenzer (z.B. END-ADD)

[2] Der Haltepunkt für END liegt hinter dem Bereichsbegrenzer, insbesondere für END-PERFORM hinter dem vollständigen PERFORM. Zusätzlich gibt es einen Haltepunkt vor END-PERFORM, und zwar am Ende eines Schleifendurchlaufs. Dieser zweite Haltepunkt wird mit PER angesprochen.

Hinweise zum symbolischen Testen von geschachtelten Programmen

- Setzen von Testpunkten
  - Paragraphen und Kapitel des inneren Programms, in dem die Unterbrechungsstelle liegt, können ohne Qualifikation angesprochen werden.
  - Auf Kapitel und Paragraphen in einem anderen Programm, das auch in einer anderen Übersetzungseinheit liegen kann, wird mit der S- und PROC-Qualifikation zugegriffen:

    %INSERT [S=program-id.]PROC=program-id-innen.paragraph [IN kapitel]
  - Die S-Qualifikation muß immer dann angegeben werden, wenn der Testpunkt in einem anderen getrennt übersetzten Programm gesetzt werden soll.
  - Ein Testpunkt am Beginn der Procedure Division des äußersten Programms kann mit einer PROG-Qualifikation gesetzt werden:

    %INSERT PROG=program-id.program-id

    oder ausgeschrieben:

    %INSERT S=program-id.PROC=program-id.program-id
  - Ein Testpunkt auf den Anfang eines inneren Programms kann, da S und PROC verschieden sind, nicht mit einer PROG-Qualifikation gesetzt werden, sondern muß wie folgt angegeben werden:

    %INSERT [S=program-id.]PROC=program-id-innen.program-id-innen
  - Namen in der aktuellen Übersetzungseinheit, die dort eindeutig sind, können auch ohne Qualifizierung angesprochen werden.
- Zugriff auf Daten
  - Mit %D werden die Daten des aktuellen geschachtelten Programms gefunden sowie Daten mit dem GLOBAL-Attribut, die nicht lokal verdeckt sind; d.h. es kann auf die gleichen Daten zugegriffen werden, auf die auch das Programm selbst an dieser Stelle zugreifen kann.
  - Mit %SD kann man die Daten aller dynamisch umgebenden Programme erhalten, entsprechend der aktuellen Aufrufhierarchie.
  - Mit der S- und PROC-Qualifikation kann man gezielt auf ein Datum eines anderen Programms zugreifen:

    %D PROC=program-id-innen.datenfeld

    Dies ist auch mit %SD ohne Qualifikation möglich, sofern das Datum in einem rufenden Programm liegt.

- Sowohl beim Zugriff auf Testpunkte als auch auf Daten gilt, daß die PROC-Qualifikation entsprechend der Programmverschachtelung mehrfach wiederholt werden kann.
- Das %TRACE-Kommando protokolliert alle durchlaufenen Anweisungen der aktuellen Csect; d.h. auch Anweisungen der gerufenen inneren Programme werden protokolliert, nicht aber die Anweisungen in getrennt übersetzten Programmen.
- Sofern beim Trace die Anweisungstypen angezeigt werden, meldet AID, wegen intern generierter Paragraphen, gelegentlich zusätzliche LABEL-Angaben.

7.2 Testhilfezeilen

Auf Quellprogrammebene bietet COBOL85 für die Diagnose von logischen Fehlern Testhilfezeilen an. Dabei handelt es sich um besonders gekennzeichnete Zeilen im Quellprogramm, die

– lediglich COBOL-Anweisungen für Testzwecke enthalten und

– bei der Übersetzung nach Bedarf als Anweisungs- oder als Kommentarzeilen behandelt werden können.

COBOL85 unterstützt die Anwendung von Testhilfezeilen durch folgende Sprachmittel (siehe [1]):

- Die WITH DEBUGGING MODE-Klausel im SOURCE-COMPUTER-Paragraphen der ENVIRONMENT DIVISION:

  Sie legt fest, wie die Testhilfezeilen vom Compiler zu behandeln sind: Wird sie angegeben, übersetzt er die Testhilfezeilen als normale Anweisungszeilen; fehlt sie, betrachtet er die Testhilfezeilen als Kommentar.

  Dieses Verfahren erlaubt es, die Testhilfezeilen nach der Testphase ungeändert im Quellprogramm zu belassen und vor der Übersetzung für den Produktiveinsatz lediglich die WITH DEBUGGING MODE-Klausel zu entfernen.

- Die Kennzeichnung von Testhilfezeilen durch ein D im Anzeigebereich (Spalte 7):

  Ein D in Spalte 7 einer Zeile legt fest, daß sie - abhängig vom Vorhandensein der WITH DEBUGGING MODE-Klausel - vom Compiler als Anweisungs- oder Kommentarzeile zu behandeln ist.

  Bei der Vereinbarung von Testhilfezeilen ist folgendes zu beachten:

  – Im Quellprogramm sind Testhilfezeilen erst nach dem OBJECT-COMPUTER-Paragraphen erlaubt.

  – Das COBOL-Quellprogramm muß sowohl mit als auch ohne Berücksichtigung der Testhilfezeilen syntaktisch korrekt sein.

8 Schnittstelle zwischen COBOL-Programmen und BS2000

Die Schnittstelle zwischen COBOL-Programmen und dem POSIX-Subsystem ist in Kap.13 dargestellt.

8.1 Ein-/Ausgabe über Systemdateien

Systemdateien sind normierte Ein-/Ausgabebereiche des Systems, denen bestimmte Endgeräte oder Dateien zugeordnet werden können. Sie stehen jeder Task ohne vorherige Vereinbarung zur Verfügung. Zu ihnen gehören

– die logischen Eingabedateien des Betriebssystems
SYSDTA und SYSIPT

– die logischen Ausgabedateien des Betriebssystems
SYSOUT, SYSLST, SYSLSTnn (nn = 01...99) und SYSOPT

8.1.1 COBOL-Sprachmittel

COBOL-Programme können Systemdateien dazu verwenden, kleine Datenmengen (z.B. Steueranweisungen) einzulesen oder auszugeben. Den Zugriff auf Systemdateien und den Bedienplatz unterstützt COBOL85 durch folgende Sprachmittel (siehe [1]):

- Die Vereinbarung programminterner Merknamen für Systemdateien im SPECIAL-NAMES-Paragraphen der ENVIRONMENT DIVISION:
Über diese Merknamen können sich Anweisungen der PROCEDURE DIVISION auf die zugeordneten Systemdateien beziehen (siehe unten). Es können unter anderem Merknamen vereinbart werden

  – für die Eingabedateien

    SYSDTA mit TERMINAL IS merkname

    SYSIPT mit SYSIPT IS merkname

– für die Ausgabedateien

SYSOUT	mit TERMINAL IS merkname
SYSLST	mit PRINTER IS merkname
SYSLSTnn	mit PRINTERnn IS merkname (nn = 01...99)
SYSOPT	mit SYSOPT IS merkname

- Die Anweisungen ACCEPT, DISPLAY und STOP literal der PROCEDURE DIVISION:

  Sie greifen auf Systemdateien bzw. auf den Bedienplatz zu, wobei im einzelnen gilt:

  – ACCEPT...FROM merkname

    liest aus der (im SPECIAL-NAMES-Paragraphen) mit merkname verknüpften **Eingabedatei**.

    Die Daten werden dabei linksbündig in der Länge des Empfangsfeldes der ACCEPT-Anweisung übertragen:
    Ist das Feld länger als der zu übertragende Wert, wird es am rechten Ende mit Leerzeichen aufgefüllt; ist es kürzer, wird der Wert bei der Übertragung rechts auf die Feldlänge abgeschnitten.

    Hat die Eingabedatei das Satzformat F (Sätze fester Länge, siehe 8.1.2), so gilt außerdem:
    Ist die Länge des Empfangsfeldes der ACCEPT-Anweisung größer als die logische Satzlänge der Systemdatei, werden automatisch Daten nachgefordert, d.h. weitere Leseoperationen (Makroaufrufe) veranlaßt.

    Erkennt das Programm beim Lesen der Systemdatei das Dateiende, gibt es die Meldung COB9121 bzw. COB9122 aus.
    Abhängig vom COMOPT-Operanden CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION in der RUNTIME-OPTIONS-Option (SDF) wird der Programmablauf anschließend fortgesetzt (Voreinstellung) oder beendet.

    Bei Fortsetzung des Programmablaufs wird im Empfangsfeld auf den ersten zwei Positionen die Zeichenfolge "/*" abgelegt (bzw. "/", wenn das Empfangsfeld nur 1 Zeichen lang ist) und mit der auf ACCEPT folgenden Anweisung fortgefahren.

  – ACCEPT (ohne FROM-Angabe)

    liest standardmäßig aus der Systemeingabedatei SYSIPT.

    Mit COMOPT REDIRECT-ACCEPT-DISPLAY=YES bzw.

    ACCEPT-DISPLAY-ASSGN=*TERMINAL in der SDF-Option RUNTIME-OPTIONS kann auf die Systemdatei SYSDTA umgewiesen werden.

– DISPLAY...UPON merkname

  schreibt in die (im SPECIAL-NAMES-Paragraphen) mit merkname verknüpfte **Ausgabedatei**.

  Die Daten werden dabei in der Länge der Sendefelder bzw. Literale der DISPLAY-Anweisung übertragen.
  Ist die Gesamtzahl der zu übertragenden Zeichen größer als die maximale Satzlänge der Ausgabedatei (siehe Tabelle 8-3), werden solange zusätzliche Datensätze ausgegeben, bis alle Zeichen übertragen sind; ist sie bei Dateien mit Sätzen fester Länge kleiner als die Satzlänge, werden die Datensätze am rechten Ende mit Leerzeichen aufgefüllt.

– DISPLAY (ohne UPON-Angabe)

  schreibt standardmäßig in die Systemausgabedatei SYSLST.

  Mit COMOPT REDIRECT-ACCEPT-DISPLAY=YES bzw.
  ACCEPT-DISPLAY-ASSGN=*TERMINAL in der SDF-Option RUNTIME-OPTIONS kann auf die Systemdatei SYSOUT umgewiesen werden.

– STOP literal

  gibt ein (maximal 122 Zeichen langes) Literal auf dem **Bedienplatz** aus.

Beispiel 8-1: Zugriff auf eine Systemdatei über einen vereinbarten Merknamen

```
IDENTIFICATION DIVISION.
    ...
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
    ...
SPECIAL-NAMES.
    SYSIPT IS SYS-EINGABE  ------------------------------------------------ (1)

    ...
  PROCEDURE DIVISION.
     ...
     ACCEPT STEUER-FELD FROM SYS-EINGABE. ------------------------------- (2)

     ...
```

(1) Für die Systemdatei SYSIPT wird der programminterne Merkname SYS-EINGABE vereinbart.

(2) ACCEPT liest (über den Merknamen SYS-EINGABE) aus SYSIPT einen Wert in das Feld STEUER-FELD.

8.1.2 Systemdateien: Primärzuweisungen, Umweisungen, Satzformate

Primärzuweisungen

Bei Taskbeginn sind die Systemdateien im BS2000 jeweils bestimmten Ein-/Ausgabegeräten zugeordnet. Diese Zuordnung, man bezeichnet sie als **Primärzuweisung**, hängt von der Art des Auftrags (Dialog- oder Stapelbetrieb) ab; die folgende Tabelle stellt die Möglichkeiten zusammen:

Systemdatei	**Primärzuweisung**	
	im Dialogbetrieb	im Stapelbetrieb
SYSDTA	Datenstation	SPOOLIN-Datei oder ENTER-Datei
SYSIPT	keine Primärzuweisung	SPOOLIN-Datei oder ENTER-Datei
SYSOUT	Datenstation	temporäre SPOOLOUT-Datei (EAM-Datei), die bei Task-Ende auf den Drucker ausgegeben und anschließend gelöscht wird
SYSLST SYSLSTnn	temporäre SPOOLOUT-Dateien (EAM-Dateien), die bei Task-Ende auf den **Drucke**r ausgegeben und anschließend gelöscht werden.	
SYSOPT	temporäre SPOOLOUT-Datei (EAM-Datei), die bei Task-Ende auf **Diskette** oder auf **Kartenstanzer** ausgegeben und anschließend gelöscht wird.	

Tabelle 8-1: Primärzuweisungen der Systemdateien

Umweisungen

Mit dem ASSIGN-*systemdatei*-Kommando kann im Verlauf einer Task die Zuordnung der Systemdateien geändert werden, d.h. sie können anderen Geräten, Systemdateien oder auch katalogisierten Dateien zugeordnet werden. Eine ausführliche Beschreibung des Kommandos findet sich in [3].

systemdatei	**Umweisung auf**	**mit dem Kommando**
SYSDTA	katalog. Plattendatei (SAM oder ISAM) oder PLAM-Bibliothek	ASSIGN-SYSDTA dateiname ASSIGN-SYSDTA *LIBRARY(bibliothek,element)
	Kartenleser	ASSIGN-SYSDTA *CARD(...)
	Diskette	ASSIGN-SYSDTA *DISKETTE(...)
SYSIPT	katalog. Plattendatei (SAM oder ISAM)	ASSIGN-SYSIPT dateiname
	Kartenleser	ASSIGN-SYSIPT *CARD(...)
SYSOUT	katalog. Plattendatei (Band oder Platte)	ASSIGN-SYSOUT dateiname (nur im Stapelbetrieb)
SYSLST SYSLSTnn	katalog. Plattendatei (SAM)	ASSIGN-SYSLST dateiname ASSIGN-SYSLST *SYSLST-NUMBER(...)
	Pseudodatei (*DUMMY)	ASSIGN-SYSLST *DUMMY
SYSOPT	katalog. Plattendatei (SAM)	ASSIGN-SYSOPT dateiname oder ASSIGN-SYSOPT dateiname, OPEN-MODE = EXTEND
	Pseudodatei (*DUMMY)	ASSIGN-SYSOPT *DUMMY

Tabelle 8-2: Umweisungen von Systemdateien

Satzformate

Die Systemdateien verarbeiten Sätze fester Länge (Satzformat F) oder Sätze variabler Länge (Satzformat V). Die folgende Tabelle gibt einen Überblick über die jeweils zulässigen Satzformate und Satzlängen.

Systemdatei	Satzformat	Satzlänge
SYSDTA	V	Bei Eingabe über Datenstation oder Plattendatei: maximal 32 Kbyte
	F	Bei Eingabe über Kartenleser: maximal 80 byte
SYSIPT	F	80 byte
SYSOUT	V	im Stapelbetrieb: maximal 132 byte (+ 1 Vorschubzeichen)
		im Dialogbetrieb: maximal 32 Kbyte
SYSLST SYSLSTnn	V	maximal 133 byte: 1 byte Steuerinformation, 132 byte Daten
SYSOPT	F	maximal 80 byte: 72 byte Daten, die Bytes 73-80 enthalten die ersten 8 Zeichen des Namens aus der PROGRAM-ID

Tabelle 8-3: Satzformate und Satzlängen der Systemdateien

8.2 Auftrags- und Benutzerschalter

Das BS2000 stellt jedem Auftrag (Task) 32 Auftragsschalter (numeriert von 0 bis 31) und jeder Benutzerkennung 32 Benutzerschalter (numeriert von 0 bis 31) zur Verfügung (siehe [3]); sie können jeweils die Zustände ON und OFF annehmen. Mit ihrer Hilfe lassen sich die Abläufe innerhalb eines Auftrags steuern bzw. mehrere Aufträge miteinander koordinieren. So verwendet man z.B.

– Auftragsschalter, wenn sich innerhalb eines Auftrags mehrere (COBOL-) Programme verständigen müssen, etwa weil der Ablauf eines Programms von den Verarbeitungsschritten eines zuvor aufgerufenen Programms abhängt

– Benutzerschalter, wenn sich mehrere Aufträge miteinander verständigen sollen. Falls Aufträge unter verschiedenen Kennungen ablaufen, können die Benutzerschalter einer Kennung zwar von Aufträgen einer anderen Kennung ausgewertet, von ihnen jedoch nicht verändert werden.

Auftrags- und Benutzerschalter können sowohl auf Betriebssystem-Ebene durch Kommandos als auch auf Programm-Ebene über COBOL-Anweisungen abgefragt und verändert werden. Den Zugriff auf Auftrags- und Benutzerschalter unterstützt COBOL85 durch folgende Sprachmittel (siehe [1]):

- Die Vereinbarung programminterner Merknamen für Auftrags- und Benutzerschalter und ihre Zustände im SPECIAL-NAMES-Paragraphen der ENVIRONMENT DIVISION.

  Über diese Merknamen können sich Anweisungen der PROCEDURE DIVISION auf die zugeordneten Schalter und deren Zustände beziehen (siehe unten). Diese Merknamen können folgendermaßen vereinbart werden:

  – Für die Auftragsschalter über die Herstellernamen TSW-0, TSW-1,..., TSW-31, wobei die zusätzlichen Angaben ON IS... und OFF IS... die Festlegung von Bedingungsnamen für den jeweiligen Schalterzustand ermöglichen.
  So lassen sich z.B. Merknamen für Auftragsschalter 17 und seine Zustände vereinbaren durch die Angaben

  ```
  TSW-17 IS merkname-17
     ON  IS schalterzustand-ein-17
     OFF IS schalterzustand-aus-17
  ```

  – Für die Benutzerschalter über die Herstellernamen USW-0, USW-1,..., USW-31, wobei die zusätzlichen Angaben ON IS... und OFF IS... die Festlegung von Bedingungsnamen für den jeweiligen Schalterzustand ermöglichen.
  So lassen sich z.B. Merknamen für Benutzerschalter 18 und seine Zustände vereinbaren durch die Angaben

  ```
  USW-18 IS merkname-18
     ON  IS schalterzustand-ein-18
     OFF IS schalterzustand-aus-18
  ```

- Die Abfrage und Veränderung von Schaltern in der PROCEDURE DIVISION:
  - Bedingungen (z.B. in der IF-, PERFORM-, EVALUATE-Anweisung) können die im SPECIAL-NAMES-Paragraphen vereinbarten Bedingungsnamen von Schalterzuständen enthalten und sie auf diese Weise für die Steuerung des Programmablaufs auswerten.
  - SET (Format 3; siehe [1]) kann über die im SPECIAL-NAMES-Paragraphen vereinbarten Merknamen auf Schalter zugreifen und ihre Zustände verändern.

Beispiel 8-2: Verwendung von Auftragsschaltern

Im folgenden Ausschnitt aus einem Dialogauftrag sieht eine DO-Prozedur verschiedene Verarbeitungsvarianten vor, die abhängig vom Zustand der Auftragsschalter 12 und 13 ausgeführt werden. Die Schalter werden sowohl auf Betriebssystem-Ebene als auch auf Programm-Ebene verändert und ausgewertet:
Zunächst kann Auftragsschalter 12 auf Betriebssystem-Ebene gesetzt werden, um die Verarbeitung innerhalb der folgenden DO-Prozedur zu steuern. Dort wird auf Programmebene sein Zustand ausgewertet und, abhängig vom Programmablauf, Auftragsschalter 13 gesetzt. Dieser wird anschließend auf Betriebssystem-Ebene ausgewertet.

```
/MODIFY-JOB-SWITCHES ON=12,OFF=13  ------------------------------- (1)
  ...
/DO PROG.SYSTEM
   Die Datei PROG.SYSTEM enthält ------------------------------- (2)
   folgende Kommandos:

   /BEGIN-PROC ...
     ...
   /START-PROGRAM PROG-1  -------------------------------------- (3)
      Ausschnitt aus PROG-1:
      ...
      SPECIAL-NAMES.            )
          TSW-12 IS SCHALTER-12 |
              ON IS EIN-12      } ------------------- (4)
          TSW-13 IS SCHALTER-13 |
              ON IS EIN-13      )
      ...
      PROCEDURE DIVISION.
          ...
          IF EIN-12 PERFORM A-PAR.  ------------------- (5)
          PERFORM B-PAR.
          ...
          IF FELD = 99 SET SCHALTER-13 TO ON. -------- (6)
          STOP RUN.
      A-PAR.
          ...
      B-PAR.
          ...
    ...
   /SKIP-COMMANDS TO-LABEL .ENDE,IF=JOB-SWITCHES (OFF=13) ------ (7)
   /START-PROGRAM PROG-2
   /.ENDE MODIFY-JOB-SWITCHES OFF=(12,13) ------------------------ (8)
   /END-PROC
/...
```

(1) Auftragsschalter 12 erhält den Status ON, 13 den Status OFF auf Betriebssystem-Ebene.

(2) Ausschnitt aus einer DO-Prozedur.

(3) Das COBOL-Programm PROG-1 wird aufgerufen.

(4) Für die Auftragsschalter 12 (TSW-12) bzw. 13 (TSW-13) werden die programminternen Namen SCHALTER-12 bzw. SCHALTER-13 vereinbart; für ihren jeweiligen ON-Status die Bedingungsnamen EIN-12 bzw. EIN-13.

(5) Falls Auftragsschalter 12 den Status ON hat (siehe (1)), wird die Anweisung PERFORM A vor PERFORM B ausgeführt.

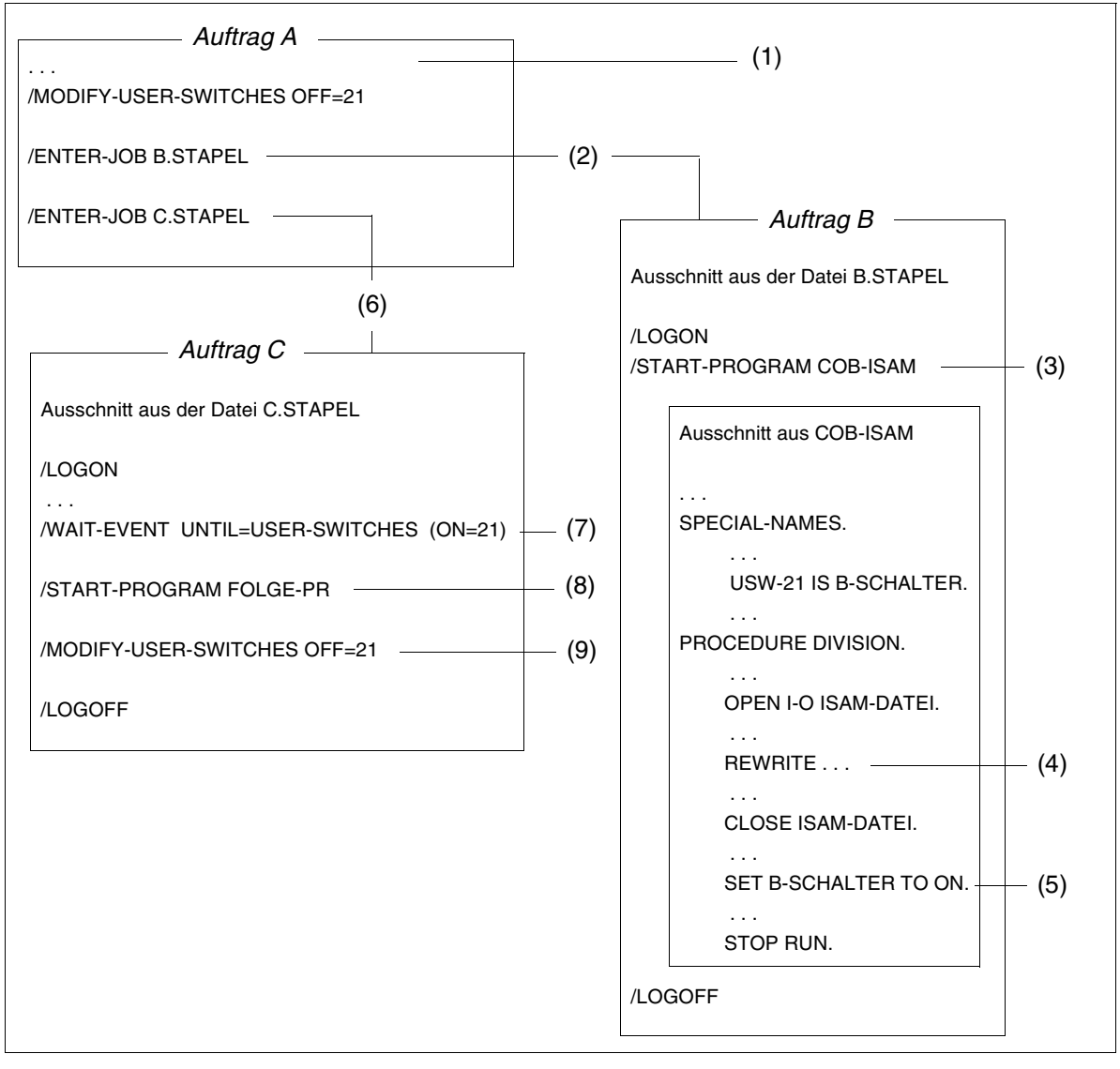
(6) Falls am Ende des Programmablaufs der Indikator FELD den Wert 99 enthält, setzt PROG-1 den Auftragsschalter 13 auf ON.

(7) Die Prozedur wertet den Status des Auftragsschalters 13 aus: Falls er von PROG-1 nicht auf ON gesetzt wurde, verzweigt sie zum Ende, andernfalls führt sie zusätzlich zu PROG-1 das Programm PROG-2 aus.

(8) Auf Betriebssystem-Ebene werden die Auftragsschalter 12 und 13 zurückgesetzt.

Beispiel 8-3: Verwendung von Benutzerschaltern

Im folgenden Ausschnitt erzeugt ein Dialogauftrag A zwei Stapelaufträge B und C. Im Auftrag B wird eine ISAM-Datei aktualisiert. Erst danach kann Auftrag C ablaufen. Benutzerschalter 21 wird in drei verschiedenen Aufträgen verwendet. Auf Programm-Ebene wird er gesetzt, auf Betriebssystem-Ebene wird er ausgewertet und rückgesetzt.



(1) Der Benutzerschalter 21 wird mit OFF initialisiert.

(2) Die ENTER-Prozedur B.STAPEL wird aufgerufen; sie erzeugt den Stapelauftrag B.

(3) Stapelauftrag B ruft das COBOL-Programm COB-ISAM auf.

(4) COB-ISAM aktualisiert die Datei ISAM-DATEI.

(5) Am Ende der Aktualisierung setzt COB-ISAM den Benutzerschalter 21 auf ON.

(6) Die ENTER-Prozedur C.STAPEL wird aufgerufen; sie erzeugt den Stapelauftrag C.

(7) Auftrag C wartet solange, bis im Auftrag B der Benutzerschalter den Status ON erhält.

(8) Sobald der Benutzerschalter 21 auf ON gesetzt ist, ruft Auftrag C das COBOL-Programm FOLGE-PR auf; es kann dann auf die im Auftrag B aktualisierte ISAM-DATEI zugreifen.

(9) Benutzerschalter 21 erhält den Zustand OFF, um das (normale) Ende von Auftrag C zu markieren.

8.3 Jobvariablen

Jobvariablen sind als eigenes Softwareprodukt erhältlich. Ähnlich wie Auftrags- und Benutzerschalter dienen auch sie dem Informationsaustausch

– zwischen Anwenderprogrammen und dem Betriebssystem oder

– zwischen verschiedenen Anwenderprogrammen.

Jobvariablen bieten jedoch gegenüber den Schaltern zusätzliche Möglichkeiten:

– Sie können beim Aufruf eines Programms als überwachende Jobvariablen vereinbart werden. Als solche werden sie vom Programm automatisch mit Zustands- und Rückkehrcodes versorgt, die über Programmzustand und Beendigungsverhalten sowie über mögliche Ablauffehler informieren.

– Sie können auf Betriebssystem- oder Programm-Ebene mit Datensätzen bis zu 256 byte (bei überwachenden Jobvariablen: 128 byte) Länge versorgt werden. Dadurch lassen sie beim Informationsaustausch eine stärkere Differenzierung zu als Auftrags- oder Benutzerschalter, die nur zwischen den Zuständen ON und OFF wechseln können.

– Sie können - anders als Auftrags- oder Benutzerschalter - auch von Aufträgen verändert werden, die unter verschiedenen Benutzerkennungen ablaufen.

Bevor ein COBOL-Programm auf eine Jobvariable zugreifen kann, muß sie ihm - ähnlich wie eine Datei - über einen Linknamen zugewiesen werden. Bei Jobvariablen dient dazu das Kommando SET-JV-LINK. Sein Format ist in den Handbüchern [3] und [7] beschrieben, ein Beispiel dazu enthält der folgende Abschnitt. Der Linkname, der dabei im Kommando anzugeben ist, ergibt sich aus den Vereinbarungen im COBOL-Programm (siehe unten).

Den Zugriff auf Jobvariablen unterstützt COBOL85 durch folgende Sprachmittel (siehe [1]):

- Die Vereinbarung von Linknamen und programminternen Merknamen für Jobvariablen im SPECIAL-NAMES-Paragraphen der ENVIRONMENT DIVISION:
  Über die Linknamen können Jobvariablen zugewiesen werden, über die Merknamen können sich die Anweisungen der PROCEDURE DIVISON auf sie beziehen (siehe unten). Link- und Merknamen für Jobvariablen lassen sich mit Angaben nach folgendem Format vereinbaren:

  JV-jvlink IS merkname

jvlink	legt dabei den Linknamen für die Jobvariable fest. Bei der Bildung des Linknamens wird vor jvlink als erstes Zeichen "*" gesetzt; er ergibt sich damit als *jvlink. Daher darf die Zeichenfolge jvlink höchstens 7 byte lang sein.
merkname	vereinbart den programminternen Merknamen für die Jobvariable.

- Die Anweisungen ACCEPT und DISPLAY der PROCEDURE DIVISION:
  - ACCEPT...FROM merkname

    liest den Inhalt der (im SPECIAL-NAMES-Paragraphen) mit merkname verknüpften Jobvariable. Die Daten werden dabei linksbündig in der Länge des Empfangsfeldes der ACCEPT-Anweisung übertragen: Ist das Feld länger als 256 byte, wird es am rechten Ende mit Leerzeichen aufgefüllt; ist es kürzer, wird der Inhalt der Jobvariable bei der Übertragung rechts auf die Feldlänge abgeschnitten.
  - DISPLAY...UPON merkname

    schreibt in die (im SPECIAL-NAMES-Paragraphen) mit merkname verknüpfte Jobvariable.
    Die Daten werden dabei in der Länge der Sendefelder bzw. Literale der DISPLAY-Anweisung übertragen, falls die maximale Datensatzlänge von 256 byte (bei überwachenden Jobvariablen: 128 byte) nicht überschritten wird. Ist die Gesamtzahl der zu übertragenden Zeichen größer als die maximale Datensatzlänge, wird der Satz bei der Übertragung auf die maximale Länge abgeschnitten.
    Bei der Übertragung in eine überwachende Jobvariable ist zu beachten, daß deren erste 128 Bytes vom System gegen Schreibzugriffe geschützt werden. Es wird daher nur der Teil des Datensatzes, der mit der Position 129 beginnt, ab Position 129 in die Jobvariable geschrieben.

  Läuft ein COBOL-Programm mit Anweisungen für Jobvariablen in einer BS2000-Installation ab, die Jobvariablen nicht unterstützt, werden diese Anweisungen nicht ausgeführt. Nach einer ACCEPT-Anweisung enthält das Empfangsfeld die Zeichen "/*" ab Spalte 1. Der erste Zugriffsversuch auf eine Jobvariable veranlaßt die Ausgabe der Meldung COB9120 nach SYSOUT.

  Ein fehlerhafter Zugriff auf eine Jobvariable in einer BS2000-Installation, die Jobvariablen unterstützt, führt zur Ausgabe der Meldung COB9197 nach SYSOUT (siehe Tabelle in Abschnitt 6.6).

Beispiel 8-4: Kommunikation über Jobvariable

Im folgenden Auftrag wird die Jobvariable KONTROLLE.ABLAUF sowohl von einem COBOL-Programm als auch auf Kommandoebene verwendet. Abhängig vom Inhalt der Jobvariable kann das Programm unterschiedliche Verarbeitungszweige durchlaufen und ggf. den Inhalt der Jobvariable aktualisieren. Auch ein anderer Auftrag - selbst unter einer anderen Benutzerkennung - kann auf diese Jobvariable zugreifen, falls sie mit dem Kommando CREATE-JV ...,USER-ACCESS=ALL-USERS katalogisiert wurde.

```
/SET-JV-LINK LINK-NAME=AKTUELL,JV-NAME=KONTROLLE.ABLAUF   ———— (1)
/START-PROGRAM PROG.ARBEIT-1

     Programmausschnitt:
     ...
     ENVIRONMENT DIVISION.
     CONFIGURATION SECTION.
     SPECIAL-NAMES.
         TERMINAL IS T
         JV-AKTUELL IS FELDJV. ——————————————————————————— (2)
          ...
     DATA DIVISION.
     WORKING-STORAGE SECTION.
     01  TAGDAT          PIC X(6). ———————————————————————— (3)
     01  INHALT-JV. ————————————————————————————————————— (4)
       05  AKT-DAT       PIC X(6).
       05  FILLER        PIC X(20).
       05  AKT-NUM       PIC 9(4).
          ...
     PROCEDURE DIVISION.
         ACCEPT INHALT-JV FROM FELDJV. ——————————————————— (5)
         ACCEPT TAGDAT FROM DATE.
         IF AKT-DAT NOT EQUAL TAGDAT ————————————————————— (6)
           PERFORM ARBEIT
             ELSE PERFORM SCHON-AKTUELL.
          ...
     ARBEIT.
          ...
         MOVE TAGDAT TO AKT-DAT.          }
         ADD 1 TO AKT-NUM.                } ———————————— (7)
         DISPLAY INHALT-JV UPON FELDJV.   }
          ...
     SCHON-AKTUELL.
         DISPLAY "ENDE AKTUALISIERUNG"
           UPON T.
          ...

/SHOW-JV JV-NAME(KONTROLLE.ABLAUF) ——————————————————————— (8)
%930629 AKTUALISIERUNG NR. 1679
```

(1) Die Jobvariable KONTROLLE.ABLAUF wird dem nachfolgend aufgerufenen COBOL-Programm PROG.ARBEIT-1 über den Linknamen *AKTUELL zugewiesen.

(2) Im SPECIAL-NAMES-Paragraphen von PROG.ARBEIT-1 werden für die Jobvariable der Linkname *AKTUELL und der (programminterne) Merkname FELDJV vereinbart.

(3) TAGDAT wird als Empfangsfeld für das Tagesdatum reserviert.

(4) Das Empfangsfeld für den Inhalt der Jobvariable wird vereinbart. Es enthält Teilfelder für die Aufnahme des letzten Aktualisierungsdatums (AKT-DAT) und eines Aktualisierungszählers (AKT-NUM).

(5) ACCEPT überträgt den Inhalt der Jobvariable FELDJV nach INHALT-JV.

(6) Abhängig davon, ob das Aktualisierungsdatum (AKT-DAT) der Jobvariable mit dem Tagesdatum (TAGDAT) übereinstimmt, werden im Programm verschiedene Verarbeitungsprozeduren durchlaufen.

(7) Am Ende der Verarbeitung werden die Felder AKT-DAT und AKT-NUM aktualisiert und mit DISPLAY INHALT-JV... in die Jobvariable zurückgeschrieben.

(8) Auf Betriebssystem-Ebene wird die Jobvariable gelesen: Sie enthält Datum und Nummer der letzten Aktualisierung.

8.4 Zugriff auf eine Umgebungsvariable

Auf eine Umgebungsvariable kann mit ACCEPT- bzw. DISPLAY-Anweisungen zugegriffen werden.
Der Name der Umgebungsvariablen wird mit Format 4 der DISPLAY-Anweisung festgelegt. Um auf den Inhalt der Umgebungsvariablen zuzugreifen, benötigt man Format 5 der ACCEPT-Anweisung.
Auf Systemebene muß die Umgebungsvariable mit einer SDF-P-Variablen eingerichtet werden.

Beispiel 8-5: Zugriff auf eine Umgebungsvariable

```
/SET-VAR TSTENV='AAAA BBB CC D'
/START-PROGRAM ...

Programmausschnitt:

IDENTIFICATION DIVISION.
...
SPECIAL-NAMES.
    ENVIRONMENT-NAME IS ENV-NAME
    ENVIRONMENT-VALUE IS ENV-VAR
    TERMINAL IS T
...
WORKING-STORAGE SECTION.
01  A   PIC X(15).
...
PROCEDURE DIVISION.
...
    DISPLAY "TSTENV" UPON ENV-NAME
    ACCEPT A FROM ENV-VAR
      ON EXCEPTION DISPLAY "ACCESS TO VARIABLE 'TSTENV' FAILED!" UPON T
                   END-DISPLAY
      NOT ON EXCEPTION DISPLAY "VALUE IS:" A UPON T
                       END-DISPLAY
    END-ACCEPT
```

Die Ausnahmebedingung tritt bei jedem fehlerhaften Zugriff ein. Ursachen für einen fehlerhaften Zugriff können z.B. sein:

– fehlendes SET-VAR-Kommando

– Inhalt der Variablen ist länger als das Empfangsfeld

8.5 Compiler- und Betriebssysteminformationen

COBOL-Programme können auf Informationen des Compilers und des Betriebssystems zugreifen. Dazu gehören Informationen über

– die Übersetzung des Quellprogramms

– die seit dem LOGON-Kommando verbrauchte CPU-Zeit

– die Task, in der das Programm abläuft, und

– die Datenstation, von der aus das Programm aufgerufen wurde.

Den Zugriff auf diese Informationen unterstützt COBOL85 durch folgende Sprachmittel:

- Die Vereinbarung programminterner Merknamen für die einzelnen Informationsarten im SPECIAL-NAMES-Paragraphen der ENVIRONMENT DIVISION:
  Über diese Merknamen kann die ACCEPT-Anweisung der PROCEDURE DIVISION auf die jeweilige Information zugreifen (siehe unten). Es können Merknamen vereinbart werden für Informationen über

  – die Übersetzung mit COMPILER-INFO IS merkname

  – die verbrauchte CPU-Zeit mit CPU-TIME IS merkname

  – den Prozeß mit PROCESS-INFO IS merkname

  – die Datenstation mit TERMINAL-INFO IS merkname

  – das Datum mit DATE-ISO4 IS merkname (mit Jahrhundert)

- Die ACCEPT-Anweisung in der PROCEDURE DIVISION:

  ACCEPT...FROM merkname

  bringt die (im SPECIAL-NAMES-Paragraphen) mit merkname verknüpften Informationen in das angegebene Empfangsfeld.

  Die Daten werden dabei linksbündig in der Länge des Empfangsfeldes der ACCEPT-Anweisung übertragen:
  Ist das Feld länger als der zu übertragende Wert, wird es am rechten Ende mit Leerzeichen aufgefüllt; ist es kürzer, wird der Wert bei der Übertragung rechts auf die Feldlänge abgeschnitten. Dies gilt nicht für CPU-TIME: Dort wird immer eine adäquate numerische Übertragung durchgeführt.
  In welcher Länge (und ggf. mit welcher Struktur) das Empfangsfeld zu vereinbaren ist, hängt von der Art der Information ab, die es aufnehmen soll. Die Formate der einzelnen Informationstypen können der Zusammenstellung im folgenden Abschnitt entnommen werden.

Inhalt und Struktur der Informationen

Die folgende Tabelle gibt Aufschluß über den Aufbau der Informationen, die einem COBOL-Programm über die Herstellernamen COMPILER-INFO, CPU-TIME, PROCESS-INFO, TERMINAL-INFO und DATE-ISO4 zur Verfügung gestellt werden.

Zeichenpositionen	Informationen für COMPILER-INFO
1-10	Name des Compilers
11-20	Version des Compilers Format: Vzz.zbzzzz z = Ziffer oder Leerzeichen b = Buchstabe oder Leerzeichen, (z.B. "V02.2A ")
21-30	Datum der Übersetzung Format: JJJJ-MM-TT (z.B. "2013-12-31")
31-38	Uhrzeit der Übersetzung Format: HH-MM-SS (z.B. "23-59-59")
39-68	Name der Übersetzungseinheit (PROGRAM-ID-Name)

	Information für CPU-TIME
PIC 9(6)V9(4)	CPU-Zeit auf zehntausendstel Sekunden genau

Zeichenpositionen	Informationen für PROCESS-INFO
1	Auftragstyp Inhalt: B für Batch, D für Dialog
2-5	TSN-Nummer
6-13	Benutzerkennung
14-21	Abrechnungsnummer
22	Privilegierungszeichen der Task Inhalt: U für Benutzer S für Systemverwalter
23-32	Betriebssystemversion Format: Vzz.zbzzzz (z.B. "V11.2 ")
33-40	Name des nächsten Rechners, an den die Datensichtstation angeschlossen ist

Zeichenpositionen	Informationen für PROCESS-INFO
41-120	Systemverwalter-Privilegien; die Felder enthalten 8 Leerzeichen, wenn das Privileg nicht vorhanden ist.
41-48	SECADM
49-56	USERADM
57-64	HSMSADM
65-72	SECOLTP
73-80	TAPEADM
81-88	SATFGMMF
89-96	NETADM
97-104	FTADM
105-112	FTACADM
113-120	TSOS

Zeichenpositionen	Informationen für TERMINAL-INFO
1-8	Stationsname
9-13	Anzahl der Zeichen pro Zeile
14-18	Anzahl der physikalischen Zeilen, die ausgegeben werden können, ohne daß die Informationsüberlaufkontrolle anspricht.
19-23	Anzahl der Zeichen, die ausgegeben werden können, ohne daß die Informationsüberlaufkontrolle anspricht.
24-27	Gerätetyp Ist ein Gerätetyp dem Laufzeitsystem nicht bekannt, enthalten diese Positionen Leerzeichen.

Zeichenpositionen	Informationen für DATE-ISO4
1-14	aktuelles Tagesdatum (einschließlich Jahrhundert JJJJ und Tageszahl NNN des laufenden Jahres) Format: JJJJ-MM-DDNNN␣

Tabelle 8-4: Struktur der Compiler- und Betriebssysteminformationen

Beispiel 8-6: Datenstrukturen für die Übernahme von Compiler- und Betriebssystem-Informationen durch die ACCEPT-Anweisung

```
*
 01  COMPILER-INFORMATION.
     02  COMPILER-NAME                    PIC X(10).
     02  COMPILER-VERSION                 PIC X(10).
     02  UEBERSETZUNGS-DATUM              PIC X(10).
     02  UEBERSETZUNGS-ZEIT               PIC X(8).
     02  PROGRAMM-NAME                    PIC X(30).
*
 01  CPU-ZEIT-IN-SEKUNDEN                 PIC 9(6)V9(4).
*
 01  PROZESS-INFORMATION.
     02  PROZESS-ART                      PIC X.
         88  BATCH-PROZESS                          VALUE "B".
         88  DIALOG-PROZESS                         VALUE "D".
     02  PROZESS-FOLGENUMMER              PIC 9(4).
     02  BENUTZERKENNUNG                  PIC X(8).
     02  ABRECHNUNGSNUMMER                PIC X(8).
     02  PRIVILEGIERUNGSKENNZEICHEN       PIC X.
         88  SYSTEMVERWALTER                        VALUE "S".
         88  BENUTZER                               VALUE "U".
     02  BETRIEBSSYSTEMVERSION            PIC X(10).
     02  PROZESSORNAME                    PIC X(8).
     02  SYSTEMVERWALTER-PRIVILEGIEN      PIC X(80).
*
 01  TERMINAL-INFORMATION.
     02  STATIONS-NAME                    PIC X(8).
     02  ZEICHEN-PRO-ZEILE                PIC 9(5).
     02  ZEILEN-PRO-SCHIRM                PIC 9(5).
     02  ZEICHEN-PRO-SCHIRM               PIC 9(5).
     02  GERAETE-TYP                      PIC X(4).
*
 01  AKTUELLES-DATUM.
     05  JAHR                             PIC X(4).
     05  FILLER                           PIC X.
     05  MONAT                            PIC X(2).
     05  FILLER                           PIC X.
     05  TAG                              PIC X(2).
     05  TAG-DES-JAHRES                   PIC X(3).
     05  FILLER                           PIC X.
```

9 Verarbeitung katalogisierter Dateien

Die Verarbeitung von POSIX-Dateien ist in Kapitel 13 beschrieben.

9.1 Grundsätzliches zum Aufbau und zur Verarbeitung katalogisierter Dateien

9.1.1 Grundbegriffe zum Aufbau von Dateien

Aus der Sicht eines COBOL-Anwenderprogramms ist eine Datei eine benannte und mit einer logischen Struktur (**Dateiorganisation**) versehene Menge von Datensätzen bestimmter **Satzformate** auf einem oder mehreren Datenträgern.
Für den Zugriff auf Dateien verwenden COBOL-Programme Funktionen des Datenverwaltungssystems (DVS), wobei die jeweilige **Zugriffsmethode des DVS** durch die Dateiorganisation festgelegt ist.
Aus der Sicht des DVS ist der Zugriff auf eine Datei stets die Übertragung von **Datenblöcken** zwischen einem peripheren Speicher und einem Teil des Hauptspeichers, dem sog. **Puffer**, den das Anwenderprogramm zur Aufnahme der Datenblöcke angelegt hat.

Dateiorganisation und Zugriffsmethode des DVS

Die Organisationsform einer Datei beschreibt deren logische Struktur und vereinbart damit die Art und Weise des Zugriffs. Sie wird bei der Dateierstellung festgelegt und kann nachträglich nicht mehr verändert werden.
COBOL kennt sequentielle, relative und indizierte Dateiorganisation. Die Möglichkeiten und Besonderheiten der einzelnen Organisationsformen werden in den Abschnitten 9.2, 9.3 und 9.4 näher erläutert. Jeder dieser Organisationsformen entspricht eine Zugriffsmethode des DVS. Die Zuordnung kann der folgenden Tabelle entnommen werden:

Organisationsform der Datei	**Zugriffsmethode des DVS**
sequentiell	SAM
relativ	ISAM/UPAM
indiziert	ISAM

Tabelle 9-1 Dateiorganisation und DVS-Zugriffsmethode

Datensätze und Satzformate

Ein (logischer) Datensatz ist die Einheit einer Datei, auf die das COBOL-Programm mit einer Ein-/Ausgabeanweisung zugreifen kann: Jede Leseoperation stellt dem Programm einen Datensatz zur Verfügung, jede Schreibanweisung erzeugt einen Datensatz in der Datei.

Die Sätze einer Datei lassen sich hinsichtlich ihres Satzformates klassifizieren. Für COBOL sind - abhängig von der Organisationsform der Datei - folgende Formate erlaubt:

– Sätze fester Länge (RECFORM=F)

  Alle Sätze einer Datei haben die gleiche Länge; sie enthalten keine Satzlängeninformation.

– Sätze variabler Länge (RECFORM=V)

  Die Sätze einer Datei können verschieden lang sein. Jeder Satz enthält die Angabe seiner Länge in seinem ersten Wort, dem sog. Satzlängenfeld.
  Im COBOL-Programm ist dieses Satzlängenfeld nicht Bestandteil der Datensatzbeschreibung, und auf seinen Inhalt kann nur dann explizit zugegriffen werden, wenn für die Datei eine RECORD-Klausel mit DEPENDING ON-Angabe vereinbart wird (siehe [1]).

– Sätze undefinierter Länge (RECFORM=U)

  Die Sätze einer Datei können verschieden lang sein, enthalten jedoch keine Angaben über ihre Satzlänge.

Datenblöcke und Puffer

Ein (logischer) Datenblock ist die Einheit einer Datei, die das DVS bei einem Dateizugriff zwischen dem peripheren Speicher und dem Hauptspeicher überträgt. Zur Aufnahme dieser Datenblöcke reserviert das Programm einen Speicherbereich in seinem Adreßraum, den sog. Puffer.

Ein logischer Block kann aus einem oder mehreren Datensätzen bestehen, ein Datensatz dagegen kann sich nicht über mehr als einen logischen Block erstrecken.

Enthält ein logischer Block mehrere Datensätze, so heißen diese Sätze geblockt. Es können nur Datensätze fester oder variabler Länge geblockt werden; für Sätze undefinierter Länge ist dies nicht möglich.

Hinsichtlich seiner Größe kann ein logischer Block und damit ein Puffer

– bei Plattendateien als Standardblock, d.h. ein physischer Block (PAM-Block) von 2048 byte oder ein ganzzahliges Vielfaches davon (bis zu 16 PAM-Blöcken) und

– bei Magnetbanddateien darüber hinaus als Nichtstandardblock einer beliebigen Länge bis zu 32767 byte vereinbart werden.

Um das Umsteigen auf zukünftige Plattenformate zu erleichtern, sollten nur geradzahlige Vielfache von 2048 byte als Blockgröße im SET-FILE-LINK-Kommando bzw. mittels der Programmangaben verwendet werden.

Ein Wert für die Puffergröße wird vom Compiler bei der Übersetzung aus den Angaben im Quellprogramm über Satz- und Blocklänge für jede Datei berechnet. Diese Voreinstellung kann bei der Zuweisung der Datei durch die Angabe des BUFFER-LENGTH-Operanden im SET-FILE-LINK-Kommando verändert werden, wobei darauf zu achten ist, daß

– der Puffer mindestens so groß sein muß wie der längste Datensatz und

– bei Verarbeitung im keylosen Format (BLKCTRL = DATA) die Verwaltungsinformationen ("Pamkey") im Puffer Platz finden (siehe 9.1.4).

Außer bei neu angelegten Dateien (OPEN OUTPUT) hat die im Katalog eingetragene Blockgröße stets Vorrang gegenüber den Blockgrößenangaben im Programm bzw. im SET-FILE-LINK-Kommando.

9.1.2 Zuweisen von katalogisierten Dateien

Für jede Datei, die ein COBOL-Programm bearbeiten soll, wird in der SELECT-Klausel (siehe [1]) ein (programminterner) Name festgelegt, auf den sich die COBOL-Anweisungen für diese Datei beziehen. Bei Programmablauf muß jedem dieser Dateinamen eine aktuelle Datei zugewiesen sein.

Diese Zuweisung läßt sich vor dem Aufruf des Programms durch ein SET-FILE-LINK- bzw. ein ASSIGN-systemdatei-Kommando herstellen. Welches der beiden Kommandos zu verwenden ist, hängt vom Eintrag in der ASSIGN-Klausel (siehe [1]) der Datei ab. Ist explizit keine Datei zugewiesen, werden Voreinstellungen des Programms wirksam, die bei der Übersetzung erzeugt wurden.
Die einzelnen Möglichkeiten der Dateizuweisung sind im folgenden zusammengestellt:

Zuweisung über das SET-FILE-LINK-Kommando

Voraussetzung dafür ist, daß in der ASSIGN-Klausel nicht der Name einer Systemdatei angegeben wurde. Die Systemdateien werden durch herstellername-1 (PRINTER) oder herstellername-2 (PRINTER01...PRINTER99, SYSIPT, SYSOPT) bezeichnet.

Die Zuweisung über das SET-FILE-LINK-Kommando kann also nur erfolgen, wenn in der ASSIGN-Klausel der Dateikettungsname (Linkname) der Datei in der Form "literal" oder "datenname" angegeben ist. Mit "literal" wird der Linkname programmstatisch angegeben. Im Datenfeld "datenname" kann der Linkname dynamisch, also während des Programmablaufs veränderbar, angegeben werden.

Um eine katalogisierte Datei zuzuweisen, muß der Anwender für diese Datei vor dem Programmaufruf ein SET-FILE-LINK-Kommando absetzen, in dessen LINK-NAME-Operanden er den vereinbarten Linknamen angibt. Mit Hilfe weiterer Operanden des SET-FILE-LINK-Kommandos können damit zugleich auch Dateimerkmale festgelegt werden.

Jeder Linkname muß den Anforderungen des BS2000 an einen Linknamen genügen (siehe dazu [4]), d.h. insbesondere,

– er muß alphanumerisch sein,

– er darf aus höchstens acht Zeichen bestehen und

– darf keine Kleinbuchstaben enthalten.

Beispiel 9-1: Zuweisung einer katalogisierten Datei über das SET-FILE-LINK-Kommando

Eintrag im FILE-CONTROL-Paragraphen des COBOL-Programms LINKLIT:	`SELECT STAMM-DATEI ASSIGN TO "STAMMLNK".`
Bei der Übersetzung erzeugter Linkname:	`STAMMLNK`

Zuweisung der Datei LAGER.BESTAND und Programmaufruf:	`/SET-FILE-LINK LINK-NAME=STAMMLNK,`
	`  FILE-NAME=LAGER.BESTAND`
	`/START-PROGRAM LINKLIT`

Bei COBOL-Programmen mit SORT (siehe Kap.10) sind folgende Linknamen für das Dienstprogramm SORT reserviert und stehen für andere Dateien nicht zur Verfügung:

MERGEnn (nn=01,...99)
SORTIN
SORTINnn (nn=01,...99)
SORTOUT
SORTWK
SORTWKn (n=1,...9)
SORTWKnn (nn=01,...99)
SORTCKPT

Ist einem internen Dateinamen mit dem Linknamen "linkname" zum Programmablauf explizit keine katalogisierte Datei zugeordnet, werden die folgenden Voreinstellungen wirksam:

– Bei einer Ausgabedatei und ENABLE-UFS-ACCESS = NO versucht das Programm auf eine katalogisierte Datei mit dem Namen aus der SELECT-Klausel zuzugreifen. Findet sich unter diesem Namen kein Katalogeintrag, schreibt das Programm in eine Datei mit dem Namen FILE.COB85.linkname, die es vorher angelegt hat.
  Bei ENABLE-UFS-ACCESS = YES schreibt das Programm unmittelbar in die Datei FILE.COB85.linkname.

– Bei einer Eingabedatei, deren SELECT-Klausel die Angabe OPTIONAL enthält (siehe auch 9.2.2), verursacht der erste Lesezugriff eine AT END-Bedingung und verzweigt zu den Prozeduren, die im Programm für diesen Fall vereinbart sind.

– Bei einer Eingabedatei (ohne OPTIONAL-Angabe in der SELECT-Klausel) und ENABLE-UFS-ACCESS = NO oder einer Ein-/Ausgabedatei versucht das Programm, auf eine katalogisierte Datei mit dem Namen aus der SELECT-Klausel zuzugreifen. Findet sich unter diesem Namen kein Katalogeintrag, wird der Ablauf mit der Fehlermeldung COB9117 unterbrochen und kann nach einer korrekten Dateizuweisung mit dem RESUME-PROGRAM-Kommando fortgesetzt werden.

Eine Dateizuweisung bleibt so lange bestehen, bis sie

– entweder explizit durch ein REMOVE-FILE-LINK-Kommando oder implizit durch das Task-Ende gelöscht oder

– durch ein nachfolgendes SET-FILE-LINK-Kommando geändert wird.

Darauf ist vor allem dann zu achten, wenn in einer Task einem programminternen Dateinamen nacheinander mehrere Dateien zugeordnet werden sollen.

Über die jeweils aktuell zugewiesenen katalogisierten Dateien informiert das SHOW-FILE-LINK-Kommando (siehe dazu [3]).

Beispiel 9-2: Änderung von Dateizuweisungen

```
/SET-FILE-LINK INOUTFIL,FILE.UPDATE.1 ------------------------------------ (1)
/START-PROGRAM AKTUELL
   ...
/SET-FILE-LINK INOUTFIL,FILE.UPDATE.2 ------------------------------------ (2)
/START-PROGRAM AKTUELL
   ...
/REMOVE-FILE-LINK INOUTFIL ----------------------------------------------- (3)
```

Das COBOL-Programm AKTUELL vereinbart für eine Ein-/Ausgabedatei den Linknamen INOUTFIL. Es soll nacheinander die katalogisierten Dateien FILE.UPDATE.1 und FILE.UPDATE.2 aktualisieren.

(1) Für die nachfolgende Verarbeitung wird dem Programm AKTUELL über den Linknamen INOUTFIL die Datei FILE.UPDATE.1 zugewiesen.

(2) Nach der Verarbeitung löst ein weiteres SET-FILE-LINK-Kommando für den Linknamen INOUTFIL die bisher gültige Dateizuordnung auf und weist als neue Datei FILE.UPDATE.2 zu.

(3) REMOVE-FILE-LINK hebt die Dateizuweisung für den Linknamen INOUTFIL auf.

Zuweisung über das ASSIGN-*systemdatei*-Kommando

Voraussetzung dafür ist, daß die ASSIGN-Klausel den Namen einer Systemdatei enthält.

Durch ein ASSIGN-*systemdatei*-Kommando für die angegebene Systemdatei kann vor dem Programmaufruf

– eine katalogisierte Datei oder

– eine andere Systemdatei

zugewiesen werden. Welche Zuordnung dabei für die jeweilige Systemdatei zulässig sind, ist der Beschreibung des ASSIGN-*systemdatei*-Kommandos in [3] zu entnehmen.

Beispiel 9-3: Zuweisung einer katalogisierten Datei über das ASSIGN-*systemdatei*-Kommando

Eintrag im FILE-CONTROL-Paragraphen des COBOL-Programms LISTPROG:	`SELECT DRUCK-DATEI ASSIGN TO PRINTER.`
Zuweisung der Datei LIST.DATEI	`/ASSIGN-SYSLST LIST.DATEI`
und Programmaufruf:	`/START-PROGRAM LISTPROG`

Wird zum Programmablauf explizit keine Datei zugewiesen, führt das Programm seine Ein-/Ausgabeoperationen auf der angegebenen Systemdatei aus.

Eine Dateizuweisung bleibt so lange bestehen, bis sie

– durch das Task-Ende gelöscht oder

– durch ein nachfolgendes ASSIGN-*systemdatei*-Kommando geändert wird.

Darauf ist vor allem dann zu achten, wenn in einer Task einem programminternen Dateinamen nacheinander mehrere Dateien zugeordnet werden sollen.

Über die jeweils aktuell zugewiesenen Dateien informiert das SHOW-SYSTEM-FILE-ASSIGNMENTS-Kommando.

9.1.3 Festlegen von Dateimerkmalen

Das SET-FILE-LINK-Kommando

Für das Einrichten von Dateien und das Festlegen von Dateimerkmalen steht im BS2000 das SET-FILE-LINK-Kommando zur Verfügung. Sein vollständiges Format und eine ausführliche Beschreibung kann in den Handbüchern [3] oder [4] nachgelesen werden.

Task File Table

Zu jeder Datei, für die ein SET-FILE-LINK-Kommando mit dem Operanden

LINK-NAME=linkname

abgesetzt wird, erzeugt das DVS unter dem Dateikettungsnamen linkname in der Task File Table (TFT) der Task einen Eintrag, der alle Dateimerkmale festhält, die im SET-FILE-LINK-Kommando explizit vereinbart wurden.

Jeder dieser Einträge bleibt so lange in der TFT gespeichert, bis er

– durch ein REMOVE-FILE-LINK-Kommando für den zugeordneten Dateikettungsnamen oder bei Task-Ende zusammen mit dem der TFT gelöscht bzw.

– durch ein neues SET-FILE-LINK-Kommando für den gleichen Dateikettungsnamen überschrieben wird.

Über den aktuellen Inhalt der TFT kann man sich mit dem Kommando SHOW-FILE-LINK-Kommando informieren.

Versucht ein COBOL-Programm, eine Datei zu eröffnen, so prüft das DVS, ob in der TFT der Linkname eingetragen ist, der für die Datei bei der Übersetzung festgelegt wurde (siehe 9.1.2). Wird ein solcher Eintrag gefunden, übernimmt das Programm die Dateimerkmale aus

– dem TFT-Eintrag unter diesem Linknamen,

– den Dateieigenschaften, die explizit oder implizit im Programm vereinbart wurden und

– dem Katalogeintrag der zugehörigen Datei.

Dabei überschreiben Angaben aus dem TFT-Eintrag (d.h. die explizit im SET-FILE-LINK-Kommando festgelegten Dateimerkmale) die Dateivereinbarungen aus dem COBOL-Programm, während aus dem Katalogeintrag lediglich Dateimerkmale übernommen werden, die weder durch das Programm noch im TFT-Eintrag festgelegt sind oder im SET-FILE-LINK-Kommando als Nulloperanden vereinbart wurden.

Beim Dateizugriff kann dieses Verfahren insbesondere dann zu Konflikten führen, wenn im SET-FILE-LINK-Kommando Dateimerkmale angegeben werden, die mit den (explizit oder implizit) im COBOL-Programm festgelegten Eigenschaften oder mit dem Katalogeintrag der zugewiesenen Datei unvereinbar sind. Dies trifft vor allem auf folgende Situationen zu:

– Widersprüchliche Angaben zur **Eröffnungsart**

COBOL-Programm	SET-FILE-LINK-Kommando
OPEN INPUT...[REVERSED]	OPEN-MODE=OUTPUT oder OPEN-MODE=EXTEND
OPEN OUTPUT	OPEN-MODE=INPUT oder OPEN-MODE=REVERSE
OPEN EXTEND	OPEN-MODE=INPUT oder OPEN-MODE=REVERSE

– Widersprüchliche Angaben zur **Organisationsform** der Datei

COBOL-Programm	SET-FILE-LINK-Kommando
ASSIGN-Klausel ORGANIZATION-Klausel	ACCESS-METHOD-Operand

– Widersprüchliche Angaben zum **Satzformat**

COBOL-Programm	SET-FILE-LINK-Kommando
RECORD-Klausel RECORDING MODE-Klausel	RECORD-FORMAT-Operand

– Widersprüchliche Angaben zur **Satzlänge**

COBOL-Programm	SET-FILE-LINK-Kommando
RECORD-Klausel Datensatzerklärung	RECORD-SIZE-Operand

– Widersprüchliche Angaben zum **Satzschlüssel**

COBOL-Programm	SET-FILE-LINK-Kommando
RECORD KEY-Klausel Datensatzerklärung	KEY-POSITION-Operand KEY-LENGTH-Operand

– Widersprüchliche Angaben zum **Plattenformat** oder **Dateiformat**

Katalogeintrag	SET-FILE-LINK-Kommando
BLK-CONTR = BUF-LEN =	BLOCK-CONTROL-INFO-Operand BUFFER-LENGTH-Operand

Beispiel 9-4:Erzeugen und Abbilden eines TFT-Eintrags
(Abbildung in BS2000/OSD V9.0)

```
/SET-FILE-LINK INOUTFIL,ISAM.UPDATE,                    }
               BUFFER-LENGTH=BY-CATALOG,                } ------------------ (1)
               SUPPORT=DISK(SHARED-UPDATE=YES)          }
/SHOW-FILE-LINK INOUTFIL,INFORMATION=ALL ------------------------------ (2)
```

```
LINK-NAME ---------- FILE-NAME ------------------------------------------------
INOUTFIL            :N:$F2190202.ISAM.UPDATE
--------------------- STATUS -------------------------------------------------
STATE      = INACTIVE    ORIGIN     = FILE
------------------- PROTECTION -----------------------------------------------
RET-PER    = *BY-PROG    PROT-LEV   = *BY-PROG
BYPASS     = *BY-PROG    DESTROY    = *BY-CAT
------------------- FILE-CONTROL-BLOCK - GENERAL ATTRIBUTES ------------------
ACC-METH   = *BY-PROG    OPEN-MODE  = *BY-PROG    REC-FORM   = *BY-PROG
REC-SIZE   = *BY-PROG    BUF-LEN    = *BY-CAT     BLK-CONTR  = *BY-PROG
F-CL-MSG   = STD         CLOSE-MODE = *BY-PROG
------------------- FILE-CONTROL-BLOCK - DISK FILE ATTRIBUTES ----------------
SHARED-UPD = YES         WR-CHECK   = *BY-PROG    IO(PERF)   = *BY-PROG
IO(USAGE)  = *BY-PROG    LOCK-ENV   = *BY-PROG
------------------- FILE-CONTROL-BLOCK - TAPE FILE ATTRIBUTES ----------------
LABEL      = *BY-PROG   (DIN-R-NUM  = *BY-PROG,   TAPE-MARK  = *BY-PROG)
CODE       = *BY-PROG    EBCDIC-TR  = *BY-PROG    F-SEQ      = *BY-PROG
CP-AT-BLIM = *BY-PROG    CP-AT-FEOV = *BY-PROG    BLOCK-LIM  = *BY-PROG
REST-USAGE = *BY-PROG    BLOCK-OFF  = *BY-PROG    TAPE-WRITE = *BY-PROG
STREAM     = *BY-PROG
------------------- FILE-CONTROL-BLOCK - ISAM FILE ATTRIBUTES ----------------
KEY-POS    = *BY-PROG    KEY-LEN    = *BY-PROG    POOL-LINK  = *BY-PROG
LOGIC-FLAG = *BY-PROG    VAL-FLAG   = *BY-PROG    PROPA-VAL  = *BY-PROG
DUP-KEY    = *BY-PROG    PAD-FACT   = *BY-PROG    READ-I-ADV = *BY-PROG
WR-IMMED   = *BY-PROG    POOL-SIZE  = *BY-PROG
---------- VOLUME ------------------------------------------------------------
DEV-TYPE   = *NONE       T-SET-NAME = *NONE
VSN/DEV    = PUBN03/D3480
```

(1) Das SET-FILE-LINK-Kommando weist der Datei ISAM.UPDATE den Linknamen INOUTFIL zu und vereinbart

- für BUFFER-LENGTH, daß dem Operanden bei der Dateieröffnung der Wert aus dem Katalogeintrag für ISAM.UPDATE zugewiesen wird, und
- SHARED-UPDATE=YES; d.h. ISAM.UPDATE soll von mehreren Benutzern simultan aktualisiert werden können.

Das DVS legt einen TFT-Eintrag unter dem Namen INOUTFIL an, in den es diese Angaben übernimmt.

(2) Das SHOW-FILE-LINK-Kommando gibt den TFT-Eintrag für INOUTFIL mit den Operandenwerten aus. Dabei sind die Werte

- BUF-LEN = *CAT und
- SHARUPD = YES

auf die Angaben im SET-FILE-LINK-Kommando zurückzuführen. Alle übrigen Operanden wurden nicht explizit vereinbart und haben die voreingestellten Werte *BY-PROG oder *NONE.

9.1.4 Platten- und Dateiformate

Plattenformate

Das BS2000 unterstützt Datenträger, die unterschiedlich formatiert sind:

– **K**ey-Datenträger für das Abspeichern von Dateien, in denen die Blockkontrollinformation in einem separaten Feld ("Pamkey") pro 2Kbyte-Datenblock steht. Diese Dateien besitzen das Blockformat PAMKEY.

– **N**on-**K**ey-Datenträger für Dateien, in denen keine separaten Pamkey-Felder existieren, sondern die Blockkontrollinformation entweder fehlt (Blockformat NO) oder im jeweiligen Datenblock untergebracht ist (Blockformat DATA).

Im BS2000 werden NK-Datenträger nach der Mindestgröße der Übertragungseinheit (Transfer Unit) unterschieden. NK2-Datenträger haben die bisherige Transfer Unit von 2KByte. NK4-Datenträger haben eine Transfer Unit von 4KByte.

Das Blockformat für eine COBOL-Datei läßt sich mit dem BLOCK-CONTROL-INFO-Operanden des SET-FILE-LINK-Kommandos bestimmen:

```
SET-FILE-LINK ...
   ,BLOCK-CONTROL-INFO = BY-PROGRAM / BY-CATALOG / WITHIN-DATA-BLOCK / PAMKEY
```

Hinweis
Dateien mit BLOCK-CONTROL-INFO = NO können mit COBOL-Programmen weder erzeugt noch gelesen werden.

Für NK-ISAM-Dateien gibt es zwei weitere Operandenwerte, nämlich:

WITHIN-DATA-2K-BLOCK / WITHIN-DATA-4K-BLOCK

Die ausführliche Beschreibung des BLOCK-CONTROL-INFO-Operanden, der verschiedenen Datei- und Datenträgerstrukturen sowie der Umstellung von K-Dateiformat auf NK-Dateiformat findet sich im Handbuch "DVS Einführung und Kommandoschnittstelle" [4].

Werden im BLOCK-CONTROL-INFO- oder im BUFFER-LENGTH-Operanden des SET-FILE-LINK-Kommandos Werte angegeben, die im Widerspruch stehen
– zum Blockformat der Datei oder
– zum Datenträger, auf dem die Datei gespeichert ist, oder
– zum erforderlichen Blockungsfaktor,

wird die Dateiverarbeitung erfolglos abgebrochen. Das Laufzeitsystem meldet dies mit dem Ein-/Ausgabe-Status (File Status) 95.

Wird für eine COBOL-Datei kein SET-FILE-LINK-Kommando verwendet, gilt die vom Systemverwalter zu treffende Voreinstellung im BLKCTRL-Operanden der CLASS2-OPTION.

K-ISAM- und NK-ISAM-Dateien

ISAM-Dateien im K-Format, die die maximale Satzlänge ausnützen, werden im NK-Format länger als der nutzbare Bereich des Datenblocks. Sie können im NK-Format behandelt werden, da das DVS Verlängerungen von Datenblöcken, sog. Überlaufblöcke, bildet.

Die Bildung von Überlaufblöcken bringt folgende Probleme mit sich:

– Die Überlaufblöcke erhöhen den Platzbedarf auf der Platte und damit die Zahl der Ein-/Ausgaben während der Dateibearbeitung.

– Der ISAM-Schlüssel darf in keinem Fall in einem Überlaufblock liegen.

Überlaufblöcke können vermieden werden, wenn man dafür sorgt, daß der längste Satz der Datei nicht länger ist, als der bei NK-ISAM-Dateien nutzbare Bereich eines logischen Blokkes.

In folgender Tabelle wird dargestellt, wie man bei ISAM-Dateien errechnen kann, wieviel Platz pro logischem Block für Datensätze zur Verfügung steht.

Dateiformat	**RECORD-FORMAT**	**maximaler nutzbarer Bereich**
K-ISAM	VARIABLE	BUF-LEN
	FIXED	BUF-LEN - (s*4) wobei s = Anzahl der Sätze pro logischem Block
NK-ISAM	VARIABLE	BUF-LEN - (n*16) - 12 - (s*2) (auf nächste durch 4 teilbare Zahl abgerundet) wobei n = Blockungsfaktor s = Anzahl der Sätze pro logischem Block
	FIXED	BUF-LEN - (n*16) - 12 - (s*2) - (s*4) (auf nächste durch 4 teilbare Zahl abgerundet) wobei n = Blockungsfaktor s = Anzahl der Sätze pro logischem Block

Tabelle 9-2: Maximal nutzbarer Blockbereich bei ISAM-Dateien

Zur Erläuterung der Formeln:

Bei RECORD-FORMAT=FIXED ist sowohl bei K- als auch bei NK-ISAM-Dateien pro Satz ein 4 byte langes Satzlängenfeld zwar vorhanden, wird aber nicht zur RECSIZE gerechnet. Deshalb müssen in diesen Fällen pro Satz jeweils 4 byte abgezogen werden.
Bei NK-ISAM-Dateien enthält jede PAM-Seite eines logischen Blocks jeweils 16 byte Verwaltungsinformation. Der logische Block enthält zusätzlich weitere 12 byte Verwaltungsinformation und pro Satz einen 2 byte langen Satzpointer.

Beispiel 9-5: maximale Satzlänge einer NK-ISAM-Datei (feste Satzlänge)

Dateivereinbarung:

```
/SET-FILE-LINK ...,RECORD-FORMAT=FIXED,BUFFER-LENGTH=STD(SIZE=2),
                                       BLOCK-CONTROL-INFO=WITHIN-DATA-BLOCK
```

maximale Satzlänge (nach Formel in Tab. 9-2):
4096 - (2*16) - 12 - 1*2 - 1*4 = 4046,
abgerundet auf die nächste durch vier teilbare Zahl: 4044 (byte).

K-SAM- und NK-SAM-Dateien

Bei SAM-Dateien gibt es keine Überlaufblöcke. Deshalb können SAM-Dateien im K-Format, die die maximale Satzlänge ausnützen, nicht in NK-SAM-Dateien umgewandelt werden. COBOL-Programme, die mit solchen für K-SAM-Dateien maximalen Satzlängen arbeiten, sind mit NK-SAM-Dateien nicht mehr ablauffähig.

In folgender Tabelle wird dargestellt, wieviel Platz bei SAM-Dateien pro logischem Block für Datensätze zur Verfügung steht.

Dateiformat	**RECORD-FORMAT**	**maximal nutzbarer Bereich**
K-SAM	VARIABLE	BUF-LEN - 4
	FIXED / UNDEFINED	BUF-LEN
NK-SAM	VARIABLE / FIXED / UNDEFINED	BUF-LEN - 16

Tabelle 9-3: Maximal nutzbarer Blockbereich bei SAM-Datei

Der Abzug von 4 byte bei K-SAM-Dateien mit variabler Satzlänge resultiert daraus, daß die logischen Blöcke solcher Dateien ein Blocklängenfeld dieser Länge enthalten, das nicht zur BUF-LEN gerechnet wird.

9.2 Sequentielle Dateiorganisation

Es gibt zwei Arten sequentiell organisierter Dateien: satzsequentielle und zeilensequentielle Dateien. Die folgende allgemeine Beschreibung bezieht sich auf satzsequentiell organisierte Dateien.
Die Abweichungen und Einschränkungen der zeilensequentiellen Organisation gegenüber der satzsequentiellen Organisation sind in Abschnitt 9.2.5 beschrieben.

9.2.1 Merkmale sequentieller Dateiorganisation

Die Sätze einer sequentiell organisierten Datei sind logisch stets in der Reihenfolge angeordnet, in der sie in die Datei geschrieben worden sind:

– Jeder Satz (außer dem letzten) hat einen eindeutigen Nachfolger und

– jeder Satz (außer dem ersten) hat einen eindeutigen Vorgänger.

Diese Vorgänger-Nachfolger-Beziehung kann während der Lebensdauer der Datei nicht geändert werden.

Es ist deshalb nicht möglich, in einer sequentiellen Datei

– Sätze einzufügen,

– Sätze zu löschen oder

– die Position eines Satzes innerhalb der festgelegten Reihenfolge zu verändern.

Sequentielle Dateiorganisation erlaubt es jedoch,

– bereits existierende Sätze zu aktualisieren
(sofern ihre Längen dabei nicht verändert werden und es sich um eine Plattenspeicherdatei handelt) und

– neue Sätze am Dateiende hinzuzufügen.

Es gibt keine Möglichkeit direkt (wahlfrei) auf jeden einzelnen Satz einer Datei zuzugreifen: Die Sätze können nur in der gleichen Reihenfolge verarbeitet werden, in der sie in der Datei stehen.

Für die Bearbeitung sequentieller Dateien verwenden COBOL-Programme die Zugriffsmethode SAM des DVS. Einzelheiten darüber können im Handbuch [4] nachgelesen werden.

Sequentielle Dateien können sowohl auf Magnetbändern als auch auf Geräten mit direktem Zugriff (Plattenspeichern) eingerichtet werden.

9.2.2 COBOL-Sprachmittel für die Verarbeitung sequentieller Dateien

Das folgende Programmskelett gibt einen Überblick über die wichtigsten Klauseln und Anweisungen, die COBOL85 für die Verarbeitung sequentieller Dateien zur Verfügung stellt. Die wesentlichen Angaben werden im Anschluß daran kurz erläutert:

```
IDENTIFICATION DIVISON.
    .
    .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT interner-dateiname
    ASSIGN TO externer-name
    ORGANIZATION IS SEQUENTIAL
    ACCESS MODE IS SEQUENTIAL
    FILE STATUS IS statusfelder.
    .
    .
DATA DIVISION.
FILE SECTION.
FD  interner-dateiname
    BLOCK CONTAINS blocklängenangabe
    RECORD satzlängenangabe
    RECORDING MODE IS satzformat
    ...
01  datensatz.
    nn feld-1              typ&länge.
    nn feld-2              typ&länge.
    ...
PROCEDURE DIVISON.
    ...
    OPEN open-modus interner-dateiname.
    ...
    WRITE datensatz.
    ...
    READ interner-dateiname
    ...
    REWRITE datensatz.
    ...
    CLOSE interner-dateiname.
    ...
    STOP RUN.
```

`SELECT interner-dateiname`

legt den Namen fest, unter dem die Datei im Quellprogramm angesprochen wird.

interner-dateiname muß ein gültiges Programmiererwort sein.

Das Format der SELECT-Klausel erlaubt auch die Angabe OPTIONAL für Eingabedateien, die beim Programmablauf nicht unbedingt vorhanden sein müssen.
Ist einem mit SELECT OPTIONAL vereinbarten Dateinamen beim Programmablauf keine Datei zugewiesen, so wird

– bei OPEN INPUT im Dialogbetrieb der Programmablauf mit der Meldung COB9117 unterbrochen und ein SET-FILE-LINK-Kommando angefordert, im Stapelbetrieb die AT END-Bedingung ausgelöst,

– bei OPEN I-O oder OPEN EXTEND eine Datei mit dem Namen FILE.COB85.linkname angelegt.

`ASSIGN TO externer-name`

gibt die Systemdatei an, die der Datei zugewiesen wird, oder legt den Linknamen fest, über den eine katalogisierte Datei zugeordnet werden kann.

externer-name muß entweder

– ein zulässiges Literal,

– ein in der DATA DIVISION definierter zulässiger Datenname oder

– ein gültiger Herstellername

aus dem Format der ASSIGN-Klausel sein (siehe [1]).

`ORGANIZATION IS SEQUENTIAL`

legt fest, daß die Datei satzsequentiell organisiert ist.

Die ORGANIZATION-Klausel kann bei satzsequentiellen Dateien entfallen, da satzsequentielle Dateiorganisation die Standardannahme des Compilers ist.

`ACCESS MODE IS SEQUENTIAL`

bestimmt, daß auf die Sätze der Datei nur sequentiell zugegriffen werden kann.
Die ACCESS MODE-Klausel ist optional und dient bei sequentiellen Dateien lediglich der Dokumentation, da sequentieller Zugriff die Standardannahme des Compilers und die einzige für sequentielle Dateien erlaubte Zugriffsart ist.

`FILE STATUS IS statusfelder`

gibt die Datenfelder an, in denen das Laufzeitsystem nach jedem Zugriff auf die Datei Informationen darüber hinterlegt,

– ob die Ein-/Ausgabeoperation erfolgreich war und

– welcher Art ggf. die dabei aufgetretenen Fehler sind.

Die statusfelder müssen in der WORKING-STORAGE SECTION oder der LINKAGE SECTION vereinbart werden. Ihr Format und die Bedeutung der einzelnen Zustandscodes werden in Abschnitt 9.2.8 beschrieben.
Die FILE STATUS-Klausel ist optional. Wird sie nicht angegeben, stehen dem Programm die oben erwähnten Informationen nicht zur Verfügung.

`BLOCK CONTAINS blocklängenangabe`

legt die maximale Größe eines logischen Blockes fest. Sie bestimmt, wie viele Datensätze jeweils gemeinsam durch eine Ein-/Ausgabeoperation in den bzw. aus dem Puffer des Programms übertragen werden sollen.

blocklängenangabe muß dabei eine zulässige Angabe aus dem Format der BLOCK CONTAINS-Klausel sein.

Die Blockung von Datensätzen verringert

– die Zahl der Zugriffe auf periphere Speicher und damit die Laufzeit des Programms und

– die Zahl der Blockzwischenräume auf dem Speichermedium und damit den physischen Platzbedarf der Datei.

Der Compiler errechnet bei der Übersetzung aus den Angaben im Quellprogramm über Block- und Satzlänge einen Wert für die Puffergröße, der bei Plattendateien vom Laufzeitsystem für das DVS auf das nächstgrößere Vielfache eines PAM-Blockes (2048 byte) aufgerundet wird. Diese Voreinstellung kann bei der Dateizuweisung durch die Angabe des BUFFER-LENGTH-Operanden im SET-FILE-LINK-Kommando verändert werden (siehe 9.1.3), wobei darauf zu achten ist, daß

– der Puffer mindestens so groß sein muß wie der längste Datensatz und

– bei Verarbeitung im keylosen Format (BLKCTRL = DATA) die Verwaltungsinformationen ("Pamkey") im Puffer Platz finden (siehe 9.1.4).

Außer bei neu angelegten Dateien (OPEN OUTPUT) hat die im Katalog eingetragene Blockgröße stets Vorrang gegenüber den Blockgrößenangaben im Programm bzw. im SET-FILE-LINK-Kommando.

Die BLOCK CONTAINS-Klausel ist optional. Wird sie nicht angegeben, nimmt der Compiler BLOCK CONTAINS 1 RECORD an, d.h. ungeblockte Datensätze.

```
RECORD satzlängenangabe
```

– legt fest, ob Sätze fester oder variabler Länge verarbeitet werden sollen und

– bestimmt bei Sätzen variabler Länge einen Bereich für die zulässigen Satzgrößen sowie, falls im Format angegeben, ein Datenfeld zur Aufnahme der jeweils aktuellen Satzlängeninformation.

satzlängenangabe muß einem der drei Formate der RECORD-Klausel entsprechen, die COBOL85 zur Verfügung stellt. Sie darf nicht im Widerspruch zu den Satzlängen stehen, die der Compiler aus den Angaben der zugehörigen Datensatzerklärung(en) errechnet.

Die RECORD-Klausel ist optional. Wird sie nicht angegeben, ergibt sich das Satzformat aus der Angabe der RECORDING MODE-Klausel (siehe unten). Fehlt auch diese, nimmt der Compiler Sätze variabler Länge an. (Zu den Abhängigkeiten zwischen RECORD- und RECORDING MODE-Klausel siehe 9.2.3).

```
RECORDING MODE IS U
```

gibt an, daß das Format der logischen Datensätze "undefiniert" ist; d.h. die Datei kann eine beliebige Kombination von festen oder variablen Datensätzen enthalten.

Die RECORDING MODE-Klausel ist optional und nur für die Vereinbarung von Datensätzen undefinierter Länge erforderlich, da die Sätze fester und variabler Länge in der RECORD-Klausel festgelegt werden (siehe 9.2.3).

```
01 datensatz.
   nn feld-1              typ&länge
   nn feld-2              typ&länge
```

stellt eine Datensatzerklärung für die zugehörige Datei dar. Sie beschreibt den logischen Aufbau von Datensätzen.

Für jede Datei ist mindestens eine Datensatzerklärung erforderlich. Werden für eine Datei mehrere Datensatzerklärungen angegeben, ist das vereinbarte Satzformat zu beachten:

– Bei Sätzen fester Länge müssen alle Satzerklärungen die gleiche Größe haben,

– bei Sätzen variabler Länge dürfen sie nicht im Widerspruch zur Satzlängenangabe der RECORD-Klausel stehen.

Die Unterteilung von datensatz in Datenfelder (feld-1, feld-2, ...) ist optional. Für typ&länge sind die erforderlichen Längen- und Formatvereinbarungen (PICTURE- und USAGE-Klauseln etc.) einzusetzen.

```
OPEN open-modus interner-dateiname
```

eröffnet die Datei in der angegebenen Eröffnungsart open-modus für die Verarbeitung. Für open-modus sind folgende Angaben möglich:

INPUT	eröffnet die Datei als Eingabedatei; sie kann nur gelesen werden
OUTPUT	eröffnet die Datei als Ausgabedatei; sie kann nur geschrieben werden.
EXTEND	eröffnet die Datei als Ausgabedatei; sie kann erweitert werden.
I-O	eröffnet die Datei als Ein-/Ausgabedatei; sie kann (Satz für Satz) gelesen, aktualisiert und zurückgeschrieben werden.

Die Angabe open-modus legt fest, mit welchen Ein-/Ausgabeanweisungen auf die Datei zugegriffen werden darf (siehe 9.2.4).

```
WRITE datensatz
READ interner-dateiname
REWRITE datensatz
```

sind Ein-/Ausgabeanweisungen für die Datei, die jeweils einen Satz

- schreiben bzw.
- lesen bzw.
- zurückschreiben

Welche dieser Anweisungen für die Datei zulässig sind, hängt von der Eröffnungsart ab, die in der OPEN-Anweisung vereinbart wird. Dieser Zusammenhang wird in 9.2.4 beschrieben.

```
CLOSE interner-dateiname
```

beendet -je nach Angabe im Format- die Verarbeitung

- der Datei (keine weitere Angabe) oder
- einer Plattenspeichereinheit (Angabe: UNIT) oder
- einer Magnetbandspule (Angabe: REEL)

und verhindert wahlweise

- ein Rückspulen des Magnetbandes (Angabe: WITH NO REWIND) oder
- ein erneutes Eröffnen der Datei (Angabe: WITH LOCK) im selben Programmablauf.

9.2.3 Zulässige Satzformate und Zugriffsarten

Satzformate

Sequentielle Dateien können Sätze fester Länge (RECFORM=F), variabler Länge (RECFORM=V) und undefinierter Länge (RECFORM=U) enthalten. Eine Blockung ist dabei nur für Sätze fester oder variabler Länge möglich.

Im COBOL-Quellprogramm wird das Format der zu verarbeitenden Sätze in der RECORD- oder der RECORDING MODE-Klausel festgelegt (siehe [1]). Welche Angaben dabei dem jeweiligen Satzformat zugeordnet sind, ist in der folgenden Tabelle zusammengestellt:

Satzformat	Angabe in der RECORD-Klausel	Angabe in der RECORDING MODE-Klausel
feste Länge	RECORD CONTAINS...CHARACTERS (Format 1)	
variable Länge	RECORD IS VARYING IN SIZE... (Format 2) oder RECORD CONTAINS...TO... (Format 3)	
undefinierte Länge	Vereinbarung mit der RECORD-Klausel nicht möglich	RECORDING MODE IS U

Tabelle 9-4: Festlegen von Satzformaten in der RECORD- oder RECORDING MODE-Klausel

Wird keine der beiden Klauseln angegeben, nimmt der Compiler Sätze variabler Länge an.

Zugriffsarten

Auf Sätze einer sequentiellen Datei kann nur sequentiell zugegriffen werden, d.h. das Programm kann sie lediglich in der Reihenfolge verarbeiten, in der sie bei der Erstellung in die Datei geschrieben worden sind.

Im COBOL-Quellprogramm wird die Zugriffsart durch die ACCESS MODE-Klausel festgelegt; für sequentielle Dateien ist ausschließlich die Angabe ACCESS MODE IS SEQUENTIAL zulässig. Da dies auch die Voreinstellung des Compilers ist, kann die ACCESS MODE-Klausel hier entfallen.

9.2.4 Eröffnungsarten und Verarbeitungsformen

Mit den Sprachmitteln eines COBOL-Programms lassen sich sequentielle Dateien
- erstellen,
- lesen,
- durch Anfügen neuer Datensätze am Dateiende erweitern und
- durch Abändern vorhandener Datensätze aktualisieren.

Welche Ein-/Ausgabeanweisungen im Programm im einzelnen für eine Datei zulässig sind, wird dabei durch ihren Eröffnungsmodus bestimmt, der in der OPEN-Anweisung angegeben wird:

OPEN OUTPUT

Als Ein-/Ausgabeanweisung ist WRITE mit folgendem Format erlaubt:

```
                          ⎧BEFORE⎫
WRITE... [FROM...] [ ⎨      ⎬ ...]
                          ⎩AFTER ⎭

        [AT END-OF-PAGE...]
        [NOT AT END-OF-PAGE...]
        [END-WRITE]
```

In diesem Modus können sequentielle Dateien (auf Platte oder Band) neu erstellt werden. Jede WRITE-Anweisung schreibt dabei einen Satz in die Datei. Hinweise zur Erzeugung von Druckerdateien sind in Abschnitt 9.2.5 zu finden.

OPEN INPUT bzw.
OPEN INPUT...REVERSED

als Ein-/Ausgabeanweisung ist READ mit folgendem Format erlaubt:

```
READ...[NEXT]
       [INTO...]
       [AT END...]
       [NOT AT END...]
       [END-READ]
```

In diesem Modus können sequentielle Dateien (von Platte oder Band) gelesen werden. Jede READ-Anweisung liest dabei einen Satz aus der Datei.

Die Angabe OPEN INPUT...REVERSED bewirkt, daß die Sätze, beginnend mit dem letzten Satz der Datei, in umgekehrter Reihenfolge gelesen werden.

`OPEN EXTEND`

Als Ein-/Ausgabeanweisung ist WRITE mit folgendem Format erlaubt:

```
                        ⎧BEFORE⎫
WRITE...[FROM...] [ ⎨      ⎬ ...]
                        ⎩AFTER ⎭
        [AT END-OF-PAGE...]
        [NOT AT END-OF-PAGE...]
        [END-WRITE]
```

In diesem Modus können am Ende einer sequentiellen Datei neue Sätze hinzugefügt werden. Bereits vorhandene Datensätze werden dabei nicht überschrieben.

`OPEN I-O`

Als Ein-/Ausgabeanweisungen sind READ und REWRITE mit folgenden Formaten erlaubt:

```
READ...[NEXT]
       [INTO...]
       [AT END...]
       [NOT AT END...]
       [END-READ]

REWRITE...[FROM...]
          [END-REWRITE]
```

In diesem Modus können die Sätze einer sequentiellen Plattendatei gelesen (READ), durch das Programm aktualisiert und anschließend wieder zurückgeschrieben werden (REWRITE). Dabei ist darauf zu achten, daß ein Satz nur dann mit REWRITE zurückgeschrieben werden kann, wenn

– er vorher durch eine erfolgreiche READ-Anweisung gelesen und

– seine Satzlänge bei der Aktualisierung nicht verändert wurde.

Die Angabe OPEN I-O ist nur für Plattendateien zulässig.

9.2.5 Zeilensequentielle Dateien

Die zeilensequentielle Organisation von COBOL-Dateien ist ein Sprachmittel des X/Open-Standards. Das entsprechende Sprachformat lautet:

```
FILE-CONTROL.
...
[ORGANIZATION IS] LINE SEQUENTIAL
...
```

Eine zeilensequentielle Datei kann im BS2000 gespeichert werden
- als katalogisierte SAM-Datei oder
- als Element einer PLAM-Bibliothek.

Damit besteht die Möglichkeit, in einem COBOL-Programm nicht nur katalogisierte Dateien, sondern auch Dateien in Form von Bibliothekselementen zu verarbeiten.
Einschränkungen gegenüber satzsequentiellen Dateien:

- Es sind nur variabel lange Sätze zulässig (RECORD-FORMAT=V).
  Gilt nicht für ENABLE-UFS-ACCESS=YES.
- Als Eröffnungsarten sind nur OPEN INPUT und OPEN OUTPUT ohne die Angaben REVERSED und NO REWIND zulässig.
- Als Ein-/Ausgabeanweisungen sind nur READ (bei OPEN INPUT) und WRITE (bei OPEN OUTPUT) zulässig.
- In der CLOSE-Anweisung ist nur die Angabe WITH LOCK zulässig.

Die Verknüpfung einer zeilensequentiellen Datei mit einer aktuellen SAM-Datei erfolgt - wie bei satzsequentiellen Dateien - mittels SET-FILE-LINK-Kommando (siehe 9.1.2).
Die Verknüpfung mit einem Bibliothekselement geschieht mit dem SDF-P-Kommando SET-VARIABLE, das folgendermaßen aufgebaut sein muß:

```
[SET-VAR] SYSIOL-name='*LIBRARY-ELEMENT(bibliothek,element[version],typ)'
```

SYSIOL-name	SDF-P-Variable. name muß der externe Name der Datei in der ASSIGN-Klausel sein.
bibliothek	Name der PLAM-Bibliothek
element	Name des Elements
version	Versionsbezeichnung. Zulässige Angaben sind: <alphanum-name 1..24> / *UPPER[-LIMIT] / *HIGH[EST-EXISTING] / *INCR[EMENT] (nur möglich beim Schreiben) Wird keine Version angegeben, wird beim Schreiben die höchste mögliche Versionsangabe generiert, beim Lesen auf die höchste vorhandene Version zugegriffen.
typ	Elementtyp. Zulässig sind S, M, J, H, P, U, F, X, R, D.

Voraussetzung für die Verarbeitung zeilensequentieller COBOL-Dateien in Bibliothekselementen ist das Vorhandensein der Bibliothek LMSLIB, die auf der TSOS-Kennung eingerichtet sein muß.
Beim Binden eines Programms, das zeilensequentielle Dateien verarbeiten soll, muß zur Befriedigung von Externverweisen zusätzlich zu CRTE die Bibliothek $LMSLIB angegeben werden.

Beispiel 9-6: Erzeugen einer zeilensequentiellen Datei in einem Bibliothekselement

Einträge im COBOL-Quellprogramm:

```
...
FILE-CONTROL.
SELECT AFILE ASSIGN TO "LIBELEM"
    ORGANIZATION IS LINE SEQUENTIAL
...
PROCEDURE DIVISION.
...
    OPEN OUTPUT AFILE.
...
```

Zuweisung von Bibliothek und Element vor Aufruf des Programms:

```
/SET-VAR SYSIOL-LIBELEM='*LIBRARY-ELEMENT(CUST.LIB,MEYER,S)'
```

Hinweis

1. Das SET-VARIABLE-Kommando kann in eine BS2000-Prozedur eingefügt werden, sofern es sich dabei um eine strukturierte SDF-P-Prozedur handelt. Die Gestaltung einer strukturierten Prozedur ist im Benutzerhandbuch zu SDF-P [30] beschrieben.
2. Die Angaben im SET-VARIABLE-Kommando innerhalb der ‘Hochkommata‘ sind ausnahmslos in Groß-Buchstaben zu schreiben.

9.2.6 Erzeugen von Druckdateien

COBOL-Sprachmittel für Druckdateien

Die Erstellung von Dateien, die auf einem Drucker ausgegeben werden sollen, wird von COBOL85 durch folgende Sprachmittel unterstützt:
- die Angabe von symbolischen Gerätenamen in der ASSIGN-Klausel
- die LINAGE-Klausel in der Dateierklärung
- Die ADVANCING- und die END-OF-PAGE-Angabe in der WRITE-Anweisung

Der Einsatz dieser Sprachmittel ist in der COBOL85-Sprachbeschreibung [1] detailliert beschrieben. Die folgende Tabelle zeigt die Verwendung der symbolischen Gerätenamen in Verbindung mit der WRITE-Anweisung und die Generierung der entsprechenden Vorschubsteuerzeichen:

symbolischer Gerätename	WRITE-Anweisung ohne ADVANCING-Angabe	WRITE-Anweisung mit ADVANCING-Angabe	Kommentar
PRINTER literal	Standardvorschub bei fehlender ADVANCING-Angabe entspricht der Angabe AFTER 1LINE; das erste Zeichen des Datensatzes steht für Benutzerdaten zur Verfügung.	Das erste Zeichen des Datensatzes steht für Benutzerdaten zur Verfügung.	Der Platz für das Vorschubzeichen wird vom Compiler reserviert und ist dem Benutzer nicht zugänglich. Für diesen Drukkertyp ist die Angabe der LINAGE-Klausel in der Dateierklärung möglich. Es sind sowohl WRITE-Anweisungen mit als auch ohne ADVANCING-Angabe für eine Datei zulässig.
PRINTER PRINTER01 - PRINTER99	wie oben	wie oben	Der Platz für das Vorschubzeichen wird vom Compiler reserviert und ist dem Benutzer nicht zugänglich. Die LINAGE-Klausel ist für diese Datei nicht erlaubt. Die Verwendung einer WRITE-Anweisung mit und ohne ADVANCING-Angabe für ein und dieselbe Datei ist nicht zulässig. Sollte dennoch dieser Fall eintreten, wird für Sätze ohne ADVANCING-Angabe ein WRITE AFTER ADVANCING 1 LINE implizit durchgeführt.
literal	Der Vorschub wird durch das erste Zeichen in jedem logischen Datensatz kontrolliert; der Benutzer muß daher vor der Ausführung jeder solchen WRITE-Anweisung das geeignete Steuerzeichen dort zur Verfügung stelllen.	Der Benutzer muß das erste Zeichen eines logischen Datensatzes reservieren; an diese Stelle wird vom Laufzeitsystem zum Programmablauf das Vorschubzeichen eingetragen. Eventuell enthaltene Benutzerdaten werden überschrieben.	Es dürfen WRITE-Anweisungen mit und ohne ADVANCING-Angabe gemischt verwendet werden. In beiden Fällen beginnt jedoch die Benutzerinformation des Drukkersatzes erst ab dem zweiten Zeichen des Datensatzes.

Tabelle 9-5: Verwendung symbolischer Gerätenamen in Verbindung mit der WRITE-Anweisung

Vorschubsteuerzeichen für Druckdateien

Bei allen Druckdateien, deren ASSIGN-Klauseln nicht die Angabe literal enthalten, wird das Steuerbyte bei der Ausführung einer WRITE-Anweisung automatisch mit einem EBCDIC-Druckervorschubzeichen versorgt, das den in der ADVANCING-Angabe gewünschten Vorschub bewirkt (siehe die beiden folgenden Tabellen). Bei fehlender ADVANCING-Angabe wird in diesen Fällen einzeiliger Vorschub angenommen. Der Platz für das Vorschubsteuerzeichen wird vom Compiler reserviert und ist dem Benutzer nicht zugänglich.

Wird für eine Datei in der ASSIGN-Klausel literal angegeben, kann das Steuerbyte auf zwei Arten mit einem Vorschubsteuerzeichen versorgt werden:

– Eine WRITE-Anweisung mit ADVANCING-Zusatz erzeugt bei ihrer Ausführung ein EBCDIC-Steuerzeichen, das den im ADVANCING-Zusatz angegebenen Vorschub bewirkt.

– Eine WRITE-Anweisung ohne ADVANCING-Zusatz versorgt das Steuerbyte nicht. Das erforderliche Vorschubsteuerzeichen muß explizit dorthin übertragen werden, bevor die Anweisung ausgeführt wird.

  Dies gibt dem Anwender die Möglichkeit, nicht nur mit den EBCDIC-Vorschubinformationen zu arbeiten, sondern im Programm davon abweichende Vorschubsteuerzeichen zu definieren - z.B. für spezielle Drucker. Welche Zeichen dabei im einzelnen zulässig sind und wie sie bei der Druckausgabe interpretiert werden, kann in den entsprechenden Druckerhandbüchern nachgelesen werden.

  Da Vorschubsteuerzeichen meistens nicht abdruckbar sind, müssen sie im Programm mit Hilfe der SYMBOLIC CHARACTERS-Klausel definiert werden, damit sie in MOVE-Anweisungen angesprochen werden können (siehe dazu Beispiel 9-7).

Je nach Ausgabeziel werden unterschiedliche Vorschubzeichen erzeugt:

	Vorschub bei Ausgabe ins BS2000	**Vorschub bei Ausgabe ins POSIX-Dateisystem**
PRINTER literal	BS2000-Vorschubzeichen gemäß Tabelle 9-6, 9-7	Vorschubzeichen und -zeilen gemäß UNIX/SINIX-Konventionen
PRINTER	wie oben	wie oben
PRINTER01-99	wie oben	nicht unterstützt
literal	wie oben	BS2000-Vorschubzeichen gem. Tabelle 9-6, 9-7

In den folgenden Tabellen sind EBCDIC-Vorschubzeichen zusammengestellt:

Vorschub um Anzahl Zeilen	Steuerzeichen für Vorschub		
	nach dem Drucken	vor dem Drucken	
		sedezimal*)	abgedruckt
1	01	40	(Leerzeichen)
2	02	41	nicht abdruckbar
3	03	42	nicht abdruckbar
. .	. .	. .	. .
11	0B	4A	c (CENT)
12	0C	4B	. (Punkt)
13	0D	4C	< (kleiner)
14	0E	4D	((Klammer)
15	0F	4E	+ (Plus)

Tabelle 9-6: EBCDIC-Steuerzeichen für Zeilenvorschub

*) Die Werte des zweiten Halbbytes sind wegen Hardwareeigenschaften um 1 kleiner als die gewünschte Zeilenzahl.

Vorschub nach Lochbandkanal*)	Steuerzeichen für Vorschub		
	nach dem Drucken	vor dem Drucken	
		sedezimal	abgedruckt
1	81	C1	A
2	82	C2	B
3	83	C3	C
4	84	C4	D
5	85	C5	E
6	86	C6	F
7	87	C7	G
8	88	C8	H
10	8A	CA	nicht abdruckbar
11	8B	CB	nicht abdruckbar

Tabelle 9-7: EBCDIC-Steuerzeichen für Vorschub nach Lochbandkanälen

*) Ein Vorschub nach Kanal 9 oder 12 ist nicht möglich, da diese Kanäle nur zur Formularende-Bestimmung dienen.

Damit beliebige sedezimale Werte (und damit auch nicht abdruckbare Vorschubsteuerzeichen) im COBOL-Quellprogramm angesprochen werden können, gestattet es der SPECIAL-NAMES-Paragraph der ENVIRONMENT DIVISION, ihnen symbolische Namen zuzuordnen (siehe [1]). Das folgende Beispiel veranschaulicht, wie auf diese Weise Vorschubsteuerzeichen definiert werden können.

Beispiel 9-7: Versorgung des Steuerbytes mit einem sedezimalen Steuerzeichen

Der sedezimale Wert 0A soll in das Steuerbyte eines Drucksatzes übertragen werden, was einen Vorschub von 10 Zeilen nach dem Drucken bewirkt.

```
IDENTIFICATION DIVISION.
    ...
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT DRUCKDATEI ASSIGN TO "AUSGABE".
 CONFIGURATION SECTION.
    ...
 SPECIAL-NAMES.
    ...
    SYMBOLIC CHARACTERS HEX-0A IS 11   ------------------------------------ (1)
     ...
  DATA DIVISION.
 FILE SECTION.
 FD  DRUCKER-DATEI
    ...                                                                  01
DRUCK-SATZ.
     02 STEUERBYTE             PIC X.
     02 DRUCK-ZEILE             PIC X(132).
    ...
  PROCEDURE DIVISION.
    ...
      MOVE "INHALT" TO DRUCK-ZEILE.
     MOVE HEX-0A TO STEUERBYTE.   ---------------------------------------- (2)
     WRITE DRUCK-SATZ.
     ...
```

(1) Dem elften Zeichen des EBCDI-Codes - es entspricht dem sedezimalen Wert 0A - wird der symbolische Name HEX-0A zugeordnet.

(2) Die MOVE-Anweisung bezieht sich auf diesen symbolischen Namen, um den sedezimalen Wert 0A in das Steuerbyte zu übertragen.

Verwendung von ASA-Vorschubsteuerzeichen

ASA-Vorschubsteuerzeichen können nur in Dateien verwendet werden, deren Zuweisung mit ASSIGN TO literal oder ASSIGN TO datenname erfolgt.
Ferner ist für die zu verarbeitende Datei folgendes SET-FILE-LINK-Kommando erforderlich:

SET-FILE-LINK dateiname, REC-FORM=(V,A)

Die unter diesen Bedingungen verwendbaren ASA-Steuerzeichen und die korrespondierenden WRITE-Anweisungen sind folgender Tabelle zu entnehmen:

ASA-Vorschubsteuerzeichen	Format der WRITE-Anweisung
+	WRITE ... BEFORE ADVANCING 0
0	WRITE ... AFTER ADVANCING 0 oder 1
–	WRITE ... AFTER ADVANCING 2
1	WRITE ... AFTER ADVANCING PAGE oder C01
2	WRITE ... AFTER ADVANCING C02
3	WRITE ... AFTER ADVANCING C03
4	WRITE ... AFTER ADVANCING C04
5	WRITE ... AFTER ADVANCING C05
6	WRITE ... AFTER ADVANCING C06
7	WRITE ... AFTER ADVANCING C07
8	WRITE ... AFTER ADVANCING C08
A	WRITE ... AFTER ADVANCING C10
B	WRITE ... AFTER ADVANCING C11

Tabelle 9-8: ASA-Vorschubsteuerzeichen und korrespondierende WRITE-Anweisungen

9.2.7 Verarbeiten von Dateien im ASCII- oder ISO-7-Bit-Code

Die Verarbeitung einer sequentiellen Datei im ASCII- bzw. ISO-7-Bit-Code unterstützt COBOL85 durch die Klauseln (siehe [1])

– ALPHABET alphabetname-1 IS STANDARD-1 für den ASCII-Code bzw.
ALPHABET alphabetname-1 IS STANDARD-2 für den ISO-7-Bit-Code
im SPECIAL-NAMES-Paragraphen der CONFIGURATION SECTION und

– CODE-SET IS alphabetname-1 in der Dateierklärung der FILE SECTION.

ASCII-Code

Die erforderlichen Angaben im COBOL-Quellprogramm zur Verarbeitung einer Datei im ASCII-Code können dem folgenden Programmskelett entnommen werden:

```
IDENTIFICATION DIVISION.
...
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
...
SPECIAL-NAMES.
...
   ALPHABET alphabetname-1 IS STANDARD-1 ------------------------------- (1)
    ...
DATA DIVISION.
FILE SECTION.
FD  datei
   CODE-SET IS alphabetname-1 ------------------------------------------- (2)
    ...
```

(1) Die ALPHABET-Klausel verknüpft die Codeart STANDARD-1 - das ist der ASCII-Code - mit dem Namen alphabetname-1.

(2) Die CODE-SET-Klausel vereinbart die mit alphabetname-1 verknüpfte Codeart als Zeichencode für die Datei.

ISO-7-Bit-Code

Für die Verarbeitung einer Datei im ISO-7-Bit-Code können im Quellprogramm Vereinbarungen analog denen für den ASCII-Code getroffen werden (siehe oben); es ist lediglich STANDARD-2 anstelle des Schlüsselwortes STANDARD-1 in der ALPHABET-Klausel anzugeben.
Bei Magnetbanddateien im ISO-7-Bit-Code gibt es darüberhinaus auch die Möglichkeit (siehe auch 9.2.8), im SET-FILE-LINK-Kommando für die Dateizuweisung SUPPORT=TAPE(CODE=ISO7) anzugeben.

9.2.8 Verarbeiten von Magnetbanddateien

Die Verarbeitung von Magnetbanddateien unterstützt COBOL85 durch folgende Sprachmittel (siehe [1]):

– Die Angaben INPUT...REVERSED und WITH NO REWIND in der OPEN-Anweisung:

  Beide Angaben bewirken, daß beim Eröffnen der Datei nicht auf den Dateianfang positioniert wird.

  INPUT...REVERSED positioniert bei der Eröffnung auf den letzten Satz der Datei und ermöglicht ein Lesen der Datensätze in umgekehrter (absteigender) Folge.

  WITH NO REWIND kann sowohl bei OPEN INPUT als auch bei OPEN OUTPUT angegeben werden und hat zur Folge, daß bei der Ausführung der OPEN-Anweisung nicht neu positioniert wird.

– Die Angaben REEL, WITH NO REWIND und FOR REMOVAL in der CLOSE-Anweisung:

  REEL ist nur erlaubt für Mehrdatenträgerdateien, d.h. Dateien, die sich über mehr als einen Datenträger (hier: Magnetbandspule) erstrecken. Die Angabe löst beim Erreichen des Spulenendes die Ausführung von Datenträgerabschluß-Operationen aus, die im einzelnen vom Eröffnungsmodus der jeweiligen Datei abhängen (siehe dazu [1], CLOSE-Anweisung). Falls zusätzlich WITH NO REWIND oder FOR REMOVAL angegeben wurde, werden bei Erreichen des Spulenendes, auch die damit verbundenen Aktionen (siehe unten) durchgeführt.

  WITH NO REWIND bewirkt, daß nach dem Abschluß der Verarbeitung einer Datei bzw. einer Spule nicht auf den Spulenanfang zurückpositioniert wird.

  FOR REMOVAL gibt an, daß die aktuelle Spule bei Erreichen des Datei- bzw. Spulenendes entladen werden soll.

Zuweisen von Magnetbanddateien

Wie Plattendateien können auch Magnetbanddateien über das SET-FILE-LINK-Kommando zugewiesen und mit Attributen versehen werden (vgl. 9.1.2 und 9.1.3). Eine ausführliche Beschreibung des Kommandoformates für Banddateien findet sich in den Handbüchern [3] und [4].

Beispiel 9-8: Zuweisen einer Banddatei

```
/SEC-RESOURCE-ALLOC,TAPE=PAR(VOL=CA176B,TYPE=T6250,ACCESS=WRITE) ——— (1)
/CREATE-FILE BESTAND.NEU,SUPPORT=TAPE(VOLUME=CA176B,DEVICE-TYPE=T6250)— (2)
/SET-FILE-LINK AUSDAT,BESTAND.NEU ——————————————————————————— (3)
/START-PROGRAM *LIB(PLAM.LIB,AKTUALISIERUNG) ———————————————————— (4)
...
/REMOVE-FILE-LINK AUSDAT,UNLOAD-RELEASED-TAPE=YES ——————————————— (5)
```

(1) Vor allem im Stapelbetrieb ist es empfehlenswert, vor der Verarbeitung die benötigten privaten Datenträger und Geräte mit SECURE-RESOURCE-ALLOCATION zu reservieren. In diesem Fall wird das Band mit der Archivnummer CA176B auf einem Bandgerät mit der Schreibdichte 6250 bpi (TYPE=T6250) mit montiertem Schreibring (ACCESS=WRITE) angefordert.

(2) Das CREATE-FILE-Kommando

- katalogisiert die Datei BESTAND.NEU als Banddatei und
- vereinbart das Datenträgerkennzeichen (VOLUME) und das Bandgerät (DEVICE-TYPE)

(3) Das SET-FILE-LINK-Kommando verknüpft den Dateinamen BESTAND.NEU mit dem Linknamen AUSDAT.

(4) START-PROGRAM ruft das Verarbeitungsprogramm auf, das als Programm unter der Elementbezeichnung AKTUALISIERUNG in der PLAM-Bibliothek PLAM.LIB gespeichert ist.

(5) Das REMOVE-FILE-LINK-Kommando nach beendeter Verarbeitung

- löst die Verknüpfung der Datei BESTAND.NEU mit dem Linknamen AUSDAT wieder und
- bewirkt, daß das Band CA176B entladen wird. Das Bandgerät wird standardmäßig wieder freigegeben.

9.2.9 Ein-/Ausgabezustände

Jeder Datei im Programm können mit der FILE STATUS-Klausel Datenfelder zugeordnet werden, in denen das Laufzeitsystem nach jedem Zugriff auf die Datei Informationen darüber hinterlegt,

– ob die Ein-/Ausgabeoperation erfolgreich war und

– welcher Art ggf. die dabei aufgetretenen Fehler sind.

Diese Informationen können z.B. in den DECLARATIVES durch USE-Prozeduren ausgewertet werden und gestatten eine Analyse von Ein-/Ausgabefehlern durch das Programm. Als Erweiterung zum COBOL-Standard bietet COBOL85 die Möglichkeit, in diese Analyse auch die Schlüssel der DVS-Fehlermeldungen einzubeziehen. Dadurch läßt sich eine feinere Differenzierung der Fehlerursachen erreichen.

Die FILE STATUS-Klausel wird im FILE-CONTROL-Paragraphen der ENVIRONMENT DIVISION angegeben; ihr Format ist (siehe [1]):

```
FILE STATUS IS datenname-1 [datenname-2]
```

Dabei müssen datenname-1 und, falls angegeben, datenname-2 in der WORKING-STORAGE SECTION oder der LINKAGE SECTION definiert sein. Für die Formate und die möglichen Werte dieser beiden Datenfelder gelten folgende Regeln:

datenname-1

– muß als zwei Byte langes numerisches oder alphanumerisches Datenfeld erklärt werden, also z.B.

```
01 datenname-1          PIC X(2).
```

– enthält nach jeder Ein-/Ausgabeoperation auf die zugeordnete Datei einen zweistelligen numerischen Zustandscode, dessen Bedeutung der Tabelle am Ende dieses Abschnitts entnommen werden kann.

datenname-2

– muß als sechs Byte langes Gruppenfeld der folgenden Struktur erklärt werden:

```
01 datenname-2.
   02 datenname-2-1       PIC 9(2) COMP.
   02 datenname-2-2       PIC X(4).
```

– dient der Aufnahme des DVS-Fehlerschlüssels (DVS-Codes) zum jeweiligen Ein-/Ausgabezustand und enthält nach jedem Zugriff auf die zugeordnete Datei einen Wert, der vom Inhalt des Feldes datenname-1 abhängt und sich aus folgender Zusammenstellung ergibt:

Inhalt von datenname-1 ungleich 0?	DVS-Code ungleich 0?	Wert von datenname-2-1	Wert von datenname-2-2
nein	nicht relevant	undefiniert	undefiniert
ja	nein	0	undefiniert
ja	ja	64	DVS-Code der zugeordneten Fehlermeldung

Die DVS-Codes und die zugeordneten Fehlermeldungen können dem Handbuch [4] entnommen werden.

Achtung

Für *zeilen*sequentielle Dateien steht nur der durch datenname-1 repräsentierte Ein-/Ausgabezustand zur Verfügung.

Die Zustandswerte und ihre Bedeutung beziehen sich i.d.R. auf *satz*sequentielle Dateien. Bei der Verarbeitung *zeilen*sequentieller Dateien müssen bezüglich der Interpretation der Zustandswerte die spezifischen Eigenheiten der zeilensequentiellen Organisation (siehe 9.2.5) berücksichtigt werden.

Tabelle 9-9: Ein-/Ausgabezustände für sequentielle Dateien

Ein-/Ausgabe-Zustand	Bedeutung
	Erfolgreiche Ausführung
00	Die Ein-/Ausgabe-Anweisung wurde erfolgreich ausgeführt. Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar.
04	Satzlängenkonflikt: Eine READ-Anweisung wurde erfolg reich ausgeführt. Die Länge des gelesenen Datensatzes liegt jedoch nicht in den Grenzen, die durch die Satzbeschreibungen der Datei festgelegt wurden.
05	Erfolgreicher OPEN INPUT/I-O/EXTEND auf eine Datei mit OPTIONAL-Angabe, die zum Zeitpunkt der Ausführung der OPEN-Anweisung nicht vorhanden war.
07	1. Erfolgreiche OPEN-Anweisung mit NO REWIND-Klausel auf eine Datei auf UNIT-RECORD-Datenträger 2. Erfolgreiche CLOSE-Anweisung mit NO REWIND-, REEL/UNIT- oder FOR REMOVAL-Klausel auf eine Datei auf UNIT-RECORD-Datenträger
	Erfolglose Ausführung: Endebedingung
10	1. Es wurde versucht, eine READ-Anweisung auszuführen. Es war jedoch kein nächster logischer Datensatz vorhanden, da das Dateiende erreicht war. 2. Es wurde zum ersten Mal versucht, eine READ-Anweisung für eine nicht vorhandene Datei mit OPTIONAL-Angabe auszuführen.
	Erfolglose Ausführung: Permanenter Fehler
30	1. Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar (der DVS-Code liefert weitere Informationen). 2. Bei zeilensequentieller Verarbeitung: erfolgloser Zugriff auf PLAM-Element
34	Es wurde versucht, außerhalb der vom System festgelegten Bereichsgrenzen einer sequentiellen Datei zu schreiben.
35	Es wurde versucht, eine OPEN-Anweisung mit INPUT-/I-O-Angabe für eine nicht vorhandene Datei auszuführen.
37	OPEN-Anweisung auf eine Datei, die auf folgende Weise nicht eröffnet werden kann: 1. OPEN OUTPUT/I-O/EXTEND auf eine schreibgeschützte Datei (Paßwort, RETENTION-PERIOD, ACCESS=READ) 2. OPEN I-O auf eine Banddatei 3. OPEN INPUT auf eine lesegeschützte Datei (Paßwort)
38	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die vorher mit der LOCK-Angabe geschlossen wurde.

Ein-/Ausgabe-Zustand	Bedeutung
39	Die OPEN-Anweisung war aus einem der folgenden Gründe erfolglos: 1. Im SET-FILE-LINK-Kommando wurden einer oder mehrere der Operanden ACCESS-METHOD, RECORD-FORMAT bzw. RECORD-SIZE mit Werten angegeben, die von den entsprechenden expliziten oder impliziten Programmangaben abweichen. 2. Bei Eingabedateien traten Satzlängenfehler auf (Katalogüberprüfung, falls RECFORM=F). 3. Die Satzlänge ist größer als die BLKSIZE im Katalog bei Eingabedateien 4. Für eine Eingabedatei stimmt der Katalogeintrag eines der Operanden FCBTYPE, RECFORM oder RECSIZE (falls RECFORM=F) nicht mit den entsprechenden expliziten oder impliziten Programmangaben bzw. mit den entsprechenden Angaben im SET-FILE-LINK-Kommando überein.
	Erfolglose Ausführung: Logischer Fehler
41	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die bereits eröffnet ist.
42	Es wurde versucht, eine CLOSE-Anweisung für eine Datei auszuführen, die nicht eröffnet ist.
43	Bei Zugriff auf eine Plattenspeicherdatei, die mit OPEN I-O eröffnet wurde Die letzte vor Ausführung einer REWRITE-Anweisung ausgeführte Ein-/Ausgabe-Anweisung war keine erfolgreich ausgeführte READ-Anweisung.
44	Überschreiten der Bereichsgrenzen: 1. Es wurde versucht, eine WRITE-Anweisung auszuführen. Die Länge des Datensatzes liegt jedoch nicht in dem für diese Datei zulässigenBereich. 2. Es wurde versucht, eine REWRITE-Anweisung auszuführen. Der zurückzuschreibende Datensatz hat jedoch nicht die gleiche Länge wie der zu ersetzende Datensatz.
46	Es wurde versucht, eine READ-Anweisung für eine Datei auszuführen, die sich im Eröffnungsmodus INPUT oder I-O befindet; ein nächster gültiger Datensatz steht aber nicht zur Verfügung. Grund: 1. Die vorhergehende READ-Anweisung war erfolglos, ohne eine Ende-Bedingung zu verursachen, oder 2. Die vorhergehende READ-Anweisung hat eine Ende-Bedingung verursacht.

Ein-/Ausgabe-Zustand	Bedeutung
47	Es wurde versucht, eine READ-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus INPUT oder I-O befindet.
48	Es wurde versucht, eine WRITE-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus OUTPUT oder EXTEND befindet.
49	Es wurde versucht, eine REWRITE-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus I-O befindet.
	Sonstige erfolglose Ausführungen
90	Systemfehler; es ist keine weitere Information über die Ursache vorhanden.
91	Systemfehler; ein Systemaufruf war nicht erfolgreich; entweder OPEN-Fehler oder kein freies Gerät; die eigentliche Ursache ist aus dem DVS-Code ersichtlich (siehe "FILE-STATUS-Klausel")
95	Unverträglichkeit zwischen den Angaben im BLOCK-CONTROL-INFO- oder BUFFER-LENGTH-Operanden des SET-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder dem Format des verwendeten Datenträgers

9.3 Relative Dateiorganisation

9.3.1 Merkmale relativer Dateiorganisation

In einer relativ organisierten Datei ist jedem Datensatz eine Nummer zugeordnet, die seine Position in der Datei angibt: Der erste Satz hat die Nummer 1, der zweite die Nummer 2 usw.

Mit Hilfe eines im Programm vereinbarten Schlüsselfeldes kann über diese relative Satznummer direkt (wahlfrei) auf jeden Satz der Datei zugegriffen werden. Zusätzlich zu den Möglichkeiten der sequentiellen Dateiorganiation gestattet dies, in einer relativen Datei

– bei der Erstellung die Datensätze wahlfrei, d.h. in beliebiger Reihenfolge, abzuspeichern,
– bei der Bearbeitung die Datensätze wahlfrei zu lesen und zu aktualisieren
– nachträglich Sätze einzufügen, sofern die dafür vorgesehene Position (relative Satznummer) noch nicht belegt ist, und
– bereits vorhandene Datensätze logisch zu löschen.

Für die Bearbeitung relativer Dateien verwenden COBOL-Programme die Zugriffsmethoden ISAM und UPAM des DVS (siehe [4]). Sie gestatten es mehreren Anwendern, gleichzeitig die Datei zu aktualisieren (siehe Abschnitt 9.5). Bestehende Dateien haben einen festgelegten FCBTYPE. Für neu zu erstellende Dateien wird stets der FCBTYPE ISAM eingesetzt, wenn nicht mit dem ACCESS-METHOD-Operanden des SET-FILE-LINK-Kommandos der FCBTYPE PAM (SAM wird mit Fehler abgewiesen) festgelegt wurde. Bei expliziter (RECORD-Klausel) variabler Satzlänge und/oder OPEN EXTEND ist nur der FCBTYPE ISAM zulässig.
Bei der Abbildung einer relativ organisierten Datei auf ISAM wird vor den Satzanfang der 8 Byte lange Satzschlüssel (sedezimal) eingeschoben. Der relative Satzschlüssel wird auf den Schlüssel der indizierten Datei abgebildet.

Relative Dateien können ausschließlich auf Plattenspeichern eingerichtet werden.

Dateistruktur

Die Beschreibung der Dateistruktur einer ISAM-Datei findet sich in Kapitel 9.4.1. Für PAM-Dateien gilt:

In ihrer logischen Struktur kann eine PAM-Datei als eine Folge von Bereichen gleicher Länge aufgefaßt werden, die jeweils einen Datensatz aufnehmen können (in PAM-Dateien sind nur Sätze fester Länge erlaubt). Jeder dieser Bereiche kann dazu über seine relative Satznummer angesprochen werden.
Bei sequentieller Erstellung einer Datei werden diese Bereiche, beginnend mit dem ersten, nacheinander mit Datensätzen gefüllt; es kann kein Bereich übersprungen werden.

Bei wahlfreier Erstellung wird jeder Datensatz in den Bereich geschrieben, mit dessen relativer Satznummer das Schlüsselfeld vor der Ausgabeanweisung versorgt wurde.
Die zugehörige Position in der Datei errechnet das Programm aus der angegebenen Satznummer und der Satzlänge. Leere Bereiche, die bei der Ausgabe übersprungen werden, legt es als Leersätze an, d.h. es reserviert Speicherbereiche in Satzlänge und versorgt jeweils das
erste Byte mit dem sedezimalen Wert FF (HIGH-VALUE) als Kennzeichen für einen Leersatz (siehe 9.3.4).

9.3.2 COBOL-Sprachmittel für die Verarbeitung relativer Dateien

Das folgende Programmskelett gibt einen Überblick über die wichtigsten Klauseln und Anweisungen, die COBOL85 für die Verarbeitung relativer Dateien zur Verfügung stellt. Die wesentlichsten Angaben werden im Anschluß daran kurz erläutert:

```
IDENTIFICATION DIVISION.
...
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT interner-dateiname
    ASSIGN TO externer-name
    ORGANIZATION IS RELATIVE
    ACCESS MODE IS zugriffsart RELATIVE KEY IS schlüssel
    FILE STATUS IS statusfelder.
    ...
DATA DIVISION.
FILE SECTION.
FD  interner-dateiname
    BLOCK CONTAINS blocklängenangabe
    RECORD CONTAINS satzlängenangabe
    ...
01  datensatz.
    nn feld-1               typ&länge.
    nn feld-2               typ&länge.
    ...
WORKING-STORAGE SECTION.
    ...
    nn schlüssel           typ&länge
    ...
PROCEDURE DIVISION.
    ...
    OPEN open-modus interner-dateiname
    ...
    START interner-dateiname
    ...
    READ interner-dateiname
    ...
    REWRITE datensatz
    ...
    WRITE datensatz
    ...
    DELETE interner-dateiname
    ...
    CLOSE interner-dateiname
    ...
    STOP RUN.
```

`SELECT interner-dateiname`

legt den Namen fest, unter dem die Datei im Quellprogramm angesprochen wird.

interner-dateiname muß ein gültiges Programmiererwort sein.
Das Format der SELECT-Klausel erlaubt auch die Angabe OPTIONAL für Eingabedateien, die beim Programmablauf nicht unbedingt vorhanden sein müssen.

Ist einem mit SELECT OPTIONAL vereinbarten Dateinamen beim Programmablauf keine Datei zugewiesen, so wird

– bei OPEN INPUT im Dialogbetrieb der Programmablauf mit der Meldung COB9117 unterbrochen und ein SET-FILE-LINK-Kommando angefordert, im Stapelbetrieb die AT END-Bedingung ausgelöst,

– bei OPEN I-O oder OPEN EXTEND eine Datei mit dem Namen FILE.COB85.linkname angelegt.

`ASSIGN TO externer-name`

gibt die Systemdatei an, die der Datei zugewiesen wird, oder legt den Linknamen fest, über den eine katalogisierte Datei zugeordnet werden kann.

externer-name muß entweder
– ein zulässiges Literal,
– ein in der DATA DIVISION definierter zulässiger Datenname oder
– ein gültiger Herstellername

aus dem Format der ASSIGN-Klausel sein (siehe [1]).

`ORGANIZATION IS RELATIVE`

legt fest, daß die Datei relativ organisiert ist.

`ACCESS MODE IS zugriffsart`

bestimmt die Art, in der auf die Sätze der Datei zugegriffen werden kann.
Für zugriffsart sind folgende Angaben möglich (siehe auch 9.3.4):

SEQUENTIAL	legt fest, daß die Sätze nur sequentiell verarbeitet werden.
RANDOM	vereinbart, daß auf die Sätze nur wahlfrei zugegriffen wird.
DYNAMIC	gestattet, daß auf die Sätze wahlweise sequentiell oder wahlfrei zugegriffen wird.

Die ACCESS MODE-Klausel ist optional. Wird sie nicht angegeben, nimmt der Compiler ACCESS MODE IS SEQUENTIAL an.

`RELATIVE KEY IS schlüssel`

gibt das Schlüsseldatenfeld zur Aufnahme der relativen Satznummern bei wahlfreiem Zugriff auf die Datensätze an.

schlüssel muß als ganzzahliges Datenfeld ohne Vorzeichen vereinbart werden. Es darf nicht Bestandteil der zugehörigen Datensatzerklärung sein.
Bei wahlfreiem Zugriff muß schlüssel vor jeder Ein-/Ausgabeanweisung mit der relativen Nummer des Satzes versorgt werden, der bearbeitet werden soll.

Die RELATIVE KEY-Angabe ist optional bei Dateien, für die ACCESS MODE IS SEQUENTIAL vereinbart wird; bei ACCESS MODE IS RANDOM oder DYNAMIC muß sie angegeben werden.

`FILE STATUS IS statusfelder`

gibt die Datenfelder an, in denen das Laufzeitsystem nach jedem Zugriff auf die Datei Informationen darüber hinterlegt,
– ob die Ein-/Ausgabeoperation erfolgreich war und
– welcher Art ggf. die dabei aufgetretenen Fehler sind.

Die Statusfelder müssen in der WORKING-STORAGE SECTION oder der LINKAGE SECTION vereinbart werden. Ihr Format und die Bedeutung der einzelnen Zustandscodes werden in Abschnitt 9.3.6 beschrieben.

Die FILE STATUS-Klausel ist optional. Wird sie nicht angegeben, stehen dem Programm die oben erwähnten Informationen nicht zur Verfügung.

`BLOCK CONTAINS blocklängenangabe`

legt die maximale Größe eines logischen Blocks fest. Sie bestimmt, wie viele Datensätze jeweils gemeinsam durch eine Ein-/Ausgabeoperation in den bzw. aus dem Puffer des Programms übertragen werden sollen.

blocklängenangabe muß eine ganze Zahl und darf nicht kleiner sein als die Satzlänge der Datei und nicht größer als 32767. Sie gibt die Größe des logischen Blocks in byte an.

Die Blockung von Datensätzen verringert

– die Zahl der Zugriffe auf periphere Speicher und damit die Laufzeit des Programms und

– die Zahl der Blockzwischenräume auf dem Speichermedium und damit den physischen Platzbedarf der Datei.

Andererseits wird bei Zugriffen mit Sperrmechanismus im Verlauf einer Simultanverarbeitung (siehe 9.5) stets der gesamte Block gesperrt, in dem sich der aktuelle Satz befindet. Ein großer Blockungsfaktor führt in diesem Fall daher zu Einbußen an Verarbeitungsgeschwindigkeit.

Der Compiler errechnet bei der Übersetzung aus den Angaben im Quellprogramm über Block- und Satzlänge einen Wert für die Puffergröße, der vom Laufzeitsystem für das DVS auf das nächstgrößere Vielfache eines PAM-Blocks (2048 byte) aufgerundet wird. Diese Voreinstellung kann bei der Dateizuweisung durch die Angabe des BUFFER-LENGTH-Operanden im SET-FILE-LINK-Kommando verändert werden, wobei darauf zu achten ist, daß

– der Puffer mindestens so groß sein muß wie der längste Datensatz und
– bei Verarbeitung im keylosen Format (BLKCTRL = DATA) die Verwaltungsinformationen ("Pamkey") im Puffer Platz finden (siehe 9.1.4).

Außer bei neu angelegten Dateien (OPEN OUTPUT) hat die im Katalog eingetragene Blockgröße stets Vorrang gegenüber den Blockgrößenangaben im Programm bzw. im SET-FILE-LINK-Kommando.

Die BLOCK CONTAINS-Klausel ist optional. Wird sie nicht angegeben, nimmt der Compiler als Blockgröße die Satzlänge der Datei an.

```
RECORD satzlängenangabe
```

– legt fest, ob Sätze fester oder variabler Länge verarbeitet werden sollen und
– bestimmt bei Sätzen variabler Länge einen Bereich für die zulässigen Satzgrößen und, falls im Format angegeben, ein Datenfeld zur Aufnahme der jeweils aktuellen Satzlängeninformation.

satzlängenangabe muß einem der drei Formate der RECORD-Klausel entsprechen, die COBOL85 zur Verfügung stellt. Sie darf nicht im Widerspruch zu den Satzlängen stehen, die der Compiler aus den Angaben der zugehörigen Datensatzerklärung(en) errechnet.

Die RECORD-Klausel ist optional. Wird sie nicht angegeben, nimmt der Compiler Sätze variabler Länge an.

```
01  datensatz.
    nn  feld-1          typ&länge
    nn  feld-2          typ&länge
```

stellt eine Datensatzerklärung für die zugehörige Datei dar. Sie beschreibt den logischen Aufbau von Datensätzen.

Für jede Datei ist mindestens eine Datensatzerklärung erforderlich. Werden für eine Datei mehrere Datensatzerklärungen angegeben, ist das vereinbarte Satzformat zu beachten:

– Bei Sätzen fester Länge müssen alle Satzerklärungen die gleiche Größe haben,
– bei Sätzen variabler Länge dürfen sie nicht im Widerspruch zur Satzlängenangabe der RECORD-Klausel stehen.

Die Unterteilung von datensatz in Datenfelder (feld-1, feld-2,...) ist optional. Für typ&länge sind die erforderlichen Längen- und Formatvereinbarungen (PICTURE- und USAGE-Klausel etc.) einzusetzen.
Das in der RELATIVE KEY-Angabe vereinbarte Schlüsseldatenfeld darf datensatz nicht untergeordnet sein.

```
nn schlüssel  typ&länge
```

vereinbart das in der RELATIVE KEY-Angabe angegebene Schlüsseldatenfeld.

Bei der Festlegung von typ&länge ist zu beachten, daß schlüssel ein ganzzahliges Datenfeld ohne Vorzeichen sein muß.
Bei wahfreiem Zugriff muß schlüssel vor jeder Ein-/Ausgabeanweisung mit der relativen Satznummer des zu bearbeitenden Satzes versorgt werden.

```
OPEN open-modus interner-dateiname
```

eröffnet die Datei in der angegebenen Eröffnungsart open-modus für die Verarbeitung. Für open-modus sind folgende Angaben möglich:

INPUT	eröffnet die Datei als Eingabedatei; sie kann nur gelesen werden.
OUTPUT	eröffnet die Datei als Ausgabedatei; sie kann nur neu geschrieben werden.
EXTEND	eröffnet die Datei als Ausgabedatei; sie kann erweitert werden.
I-O	eröffnet die Datei als Ein-/Ausgabedatei; sie kann (Satz für Satz) gelesen, aktualisiert und zurückgeschrieben werden.

Die Angabe für open-modus legt fest, mit welchen Ein-/Ausgabeanweisungen auf die Datei zugegriffen werden darf (siehe 9.3.4).

```
START interner-dateiname
READ interner-dateiname
REWRITE datensatz
WRITE datensatz
DELETE interner-dateiname
```

sind Ein-/Ausgabeanweisungen für die Datei, die jeweils
- in der Datei auf einen Satz positionieren bzw.
- einen Satz lesen bzw.
- einen Satz zurückschreiben bzw.
- einen Satz schreiben bzw.
- einen Satz löschen.

Welche dieser Anweisungen für die Datei zulässig sind, hängt von der Eröffnungsart ab, die in der OPEN-Anweisung vereinbart wird. Dieser Zusammenhang wird in 9.3.4 beschrieben.

```
CLOSE interner-dateiname
```

beendet die Verarbeitung der Datei.

Durch die zusätzliche Angabe WITH LOCK kann ein erneutes Eröffnen der Datei im selben Programmlauf verhindert werden.

9.3.3 Zulässige Satzformate und Zugriffsarten

Satzformate

Relative Dateien können Sätze fester Länge (RECFORM=F) oder variabler Länge (RECFORM=V) enthalten. In beiden Fällen können die Sätze geblockt oder ungeblockt vorliegen.
Im COBOL-Qellprogramm kann das Format der zu verarbeitenden Sätze in der RECORD-Klausel vereinbart werden. Welche Angaben dabei dem jeweiligen Satzformat zugeordnet sind, ist in der folgenden Tabelle zusammengestellt:

Satzformat	**Angabe in der RECORD-Klausel**
Sätze fester Länge	RECORD CONTAINS...CHARACTERS (Format 1)
Sätze variabler Länge	RECORD IS VARYING IN SIZE... (Format 2) oder RECORD CONTAINS...TO... (Format 3)

Tabelle 9-10: Satzformat und RECORD-Klausel

Zugriffsarten

Auf Sätze einer relativen Datei kann sequentiell, wahlfrei oder dynamisch zugegriffen werden.

Im COBOL- Quellprogramm wird die Zugriffsart durch die ACCESS MODE-Klausel festgelegt. Die folgende Übersicht stellt die möglichen Angaben und ihre Auswirkungen auf die Zugriffsart zusammen:

Angabe in der ACCESS MODE-Klausel	**Zugriffsart**
SEQUENTIAL	Sequentieller Zugriff: Die Datensätze können nur in der Reihenfolge verarbeitet werden, in der sie entsprechend ihrer relativen Satznummer in der Datei vorkommen. Das bedeutet: Beim Lesen wird jeweils der nächste logische Datensatz zur Verfügung gestellt, wobei mögliche vorangehende Leersätze übersprungen werden. Beim Schreiben wird jeder Satz mit der nachfolgenden relativen Satznummer in die Datei ausgegeben; es werden keine Leersätze geschrieben.
RANDOM	Wahlfreier Zugriff: Die Datensätze können in beliebiger Reihenfolge über ihre relativen Satznummern angesprochen werden. Dazu muß vor jeder Ein-/Ausgabeanweisung für einen Satz dessen Nummer im RELATIVE KEY-Schlüsselfeld zur Verfügung gestellt werden.
DYNAMIC	Dynamischer Zugriff: Auf die Datensätze kann sowohl sequentiell als auch wahlfrei zugegriffen werden. Die jeweilige Zugriffsart wird dabei über das Format der Ein-/Ausgabeanweisung gewählt.

Tabelle 9-11: ACCESS MODE-Klausel und Zugriffsart

9.3.4 Eröffnungsarten und Verarbeitungsformen

Mit den Sprachmitteln eines COBOL-Programms lassen sich relative Dateien
– erstellen,
– lesen,
– durch Hinzufügen neuer Datensätze erweitern und
– durch Abändern oder Löschen vorhandener Datensätze aktualisieren.

Welche Ein-/Ausgabeanweisungen im Programm jeweils für eine Datei zulässig sind, wird dabei durch ihren Eröffnungsmodus bestimmt, der in der OPEN-Anweisung angegeben wird:

```
OPEN OUTPUT
```

Als Ein-/Ausgabeanweisung ist unabhängig von der Angabe in der ACCESS MODE-Klausel WRITE mit folgendem Format erlaubt:

```
WRITE...[FROM...]
        [INVALID KEY...]
        [NOT INVALID KEY...]
        [END WRITE...]
```

In diesem Modus können relative Dateien ausschließlich neu erstellt (geladen) werden. Abhängig von der vereinbarten Zugriffsart hat die WRITE-Anweisung dabei folgende Wirkung:

– ACCESS MODE IS SEQUENTIAL

  erlaubt, eine relative Datei sequentiell zu erstellen. WRITE schreibt dabei - beginnend mit 1 - die Sätze mit lückenlos aufsteigenden relativen Satznummern in die Datei.
  Das RELATIVE KEY-Schlüsselfeld - wenn angegeben - wird von WRITE nicht ausgewertet; es enthält jeweils die (automatisch hochgezählte) relative Satznummer des zuletzt geschriebenen Satzes.
  Da keine Leersätze erzeugt werden, ist es nicht möglich, in eine sequentiell erzeugte Datei nachträglich Datensätze einzufügen.

– ACCESS MODE IS RANDOM oder DYNAMIC

  (beide Angaben haben hier gleiche Bedeutung) ermöglicht es, eine Datei wahlfrei zu erstellen. WRITE schreibt dabei jeden Datensatz an die Position in der Datei, die dessen Satznummer angibt.
  Das RELATIVE KEY-Schlüsselfeld muß daher vor jeder WRITE-Anweisung mit der relativen Satznummer versorgt werden, die der zu schreibende Satz in der Datei erhalten soll. Wird dabei die Nummer eines bereits existierenden Satzes angegeben, tritt eine INVALID KEY-Bedingung auf und WRITE verzweigt zur INVALID-KEY-Anweisung bzw. zur vereinbarten USE-Prozedur, ohne den Satz zu schreiben. Ein Überschreiben von Datensätzen ist hier also nicht möglich.

`OPEN EXTEND`

Mit OPEN EXTEND kann eine vorhandene Datei erweitert werden. Der Zugriff kann nur sequentiell erfolgen.

– ACCESS MODE IS SEQUENTIAL

  erlaubt, eine relative Datei sequentiell zu erweitern. WRITE schreibt dabei - beginnend mit dem höchsten Schlüssel+1 - die Sätze mit lückenlos aufsteigenden relativen Satznummern in die Datei.
  Das RELATIVE KEY-Schlüsselfeld - wenn angegeben - wird von WRITE nicht ausgewertet; es enthält jeweils die (automatisch hochgezählte) relative Satznummer des zuletzt geschriebenen Satzes.
  Da keine Leersätze erzeugt werden, ist es nicht möglich, in eine sequentiell erweiterte Datei nachträglich Datensätze einzufügen.

`OPEN INPUT`

Welche Ein-/Ausgabeanweisungen bzw. Anweisungsformate erlaubt sind, hängt von der Angabe in der ACCESS MODE-KLausel ab. Die folgende Tabelle stellt die Möglichkeiten für OPEN INPUT zusammen:

Anweisung	**Eintrag in der ACCESS MODE-Klausel**		
	SEQUENTIAL	RANDOM	DYNAMIC
START	START... [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-START]	Anweisung nicht zulässig	START... [KEY IS...] [INVALID KEY...] [NOT INVALID KEY] [END-START]
READ	READ...[NEXT] [INTO...] [AT END...] [NOT AT END...] [END-READ...]	READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]	Für sequentiellen Zugriff: READ...NEXT [INTO...] [AT END...] [NOT AT END...] [END-READ]
			Für wahlfreien Zugriff: READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]

Tabelle 9-12: Erlaubte Ein-/Ausgabeanweisung für OPEN INPUT

In diesem Modus können relative Dateien gelesen werden. Abhängig von der vereinbarten Zugriffsart hat die READ-Anweisung dabei folgende Wirkung:

– ACCESS MODE IS SEQUENTIAL

  erlaubt es ausschließlich, die Datei sequentiell zu lesen. READ stellt dabei die Datensätze in der Reihenfolge aufsteigender relativer Satznummern zur Verfügung, wobei Leersätze übersprungen werden.
  Das RELATIVE KEY-Schlüsselfeld - wenn angegeben - wird von READ nicht ausgewertet; es enthält jeweils die relative Satznummer des zuletzt gelesenen Satzes.
  Falls ein RELATIVE KEY-Schlüsselfeld vereinbart wird, kann jedoch vor der Ausführung einer READ-Anweisung mit Hilfe von START auf einen beliebigen Satz der Datei positioniert werden: Über eine Vergleichsbedingung legt START die relative Satznummer des zuerst zu lesenden Satzes und damit den Ausgangspunkt für nachfolgende sequentielle Leseoperationen fest.
  Kann die Vergleichsbedingung von keiner relativen Satznummer der Datei erfüllt werden, tritt eine INVALID KEY-Bedingung auf und START verzweigt zur INVALID KEY-Anweisung bzw. zur vereinbarten USE-Prozedur.

– ACCESS MODE IS RANDOM

  ermöglicht es, die Sätze der Datei wahlfrei zu lesen. READ stellt dabei die Datensätze in beliebiger Reihenfolge zur Verfügung; der Zugriff auf jeden Satz erfolgt über seine relative Satznummer.
  Das RELATIVE KEY-Schlüsselfeld muß dazu vor jeder READ-Anweisung mit der relativen Nummer des Satzes versorgt werden, der gelesen werden soll. Wird dabei die Nummer eines nicht existierenden Satzes (z.B. eines Leersatzes) angegeben, tritt eine INVALID KEY-Bedingung auf und READ verzweigt zur INVALID KEY-Anweisung bzw. zur vereinbarten USE-Prozedur.

– ACCESS MODE IS DYNAMIC

  gestattet es, die Datei sowohl sequentiell als auch wahlfrei zu lesen.
  Die jeweilige Zugriffsart wird dabei über das Format der READ-Anweisung gewählt (siehe Tabelle 9-12).
  Eine START-Anweisung ist nur für sequentielles Lesen sinnvoll.

`OPEN I-O`

Welche Ein-/Ausgabeanweisungen bzw. Anweisungsformate erlaubt sind, hängt von der Angabe in der ACCESS MODE-Klausel ab. Die folgende Tabelle stellt die Möglichkeiten für OPEN I-O zusammen:

Anweisung	**Eintrag in der ACCESS MODE-Klausel**		
	SEQUENTIAL	RANDOM	DYNAMIC
START	START... [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-START]	Anweisung nicht zulässig	START... [KEY IS...] [INVALID KEY...] [NOT INVALID KEY] [END-START]
READ	READ...[NEXT] [INTO...] [AT END...] [NOT AT END...] [END-READ...]	READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]	Für sequentiellen Zugriff: READ...NEXT [INTO...] [AT END...] [NOT AT END...] [END-READ]
			Für wahlfreien Zugriff: READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]
REWRITE	REWRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-REWRITE]	REWRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-REWRITE]	REWRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-REWRITE]
WRITE	Anweisung nicht zulässig	WRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-WRITE]	WRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-WRITE]
DELETE	DELETE... [END-DELETE]	DELETE... [INVALID KEY...] [NOT INVALID KEY...] [END-DELETE]	DELETE... [INVALID KEY...] [NOT INVALID KEY...] [END-DELETE]

Tabelle 9-13: Erlaubte Ein-/Ausgabeanweisungen für OPEN I-O

In diesem Modus können in einer relativen Datei Sätze
- gelesen,
- hinzugefügt,
- durch das Programm aktualisiert und
- überschrieben oder
- gelöscht werden.

OPEN I-O setzt voraus, daß die zu verarbeitende Datei bereits existiert. Es ist daher nicht möglich, in diesem Modus eine relative Datei neu zu erstellen.

Welche dieser Verarbeitungsformen durchgeführt werden können, und wie die Ein-/Ausgabeanweisungen dabei wirken, hängt von der vereinbarten Zugriffsart ab:

– ACCESS MODE IS SEQUENTIAL

  erlaubt es, wie bei OPEN INPUT die Datei mit READ sequentiell zu lesen und dabei durch einen vorhergehenden START auf einen beliebigen Satz der Datei als Anfangspunkt zu positionieren.

  Darüberhinaus kann nach einem erfolgreichen READ der gelesene Satz durch das Programm aktualisiert und mit REWRITE zurückgeschrieben oder mit DELETE logisch gelöscht werden. Dabei darf zwischen READ und REWRITE bzw. DELETE keine weitere Ein-/Ausgabeanweisung für diese Datei ausgeführt werden.

– ACCESS MODE IS RANDOM

  ermöglicht es, wie bei OPEN INPUT mit READ Sätze wahlfrei zu lesen.

  Ferner können mit WRITE neue Sätze in die Datei eingefügt und mit REWRITE bzw. DELETE bereits in der Datei vorhandene Sätze überschrieben bzw. gelöscht werden (unabhängig davon, ob sie vorher gelesen wurden).

  Das RELATIVE KEY-Schlüsselfeld muß dazu vor jeder WRITE-, REWRITE- oder DELETE-Anweisung mit der relativen Nummer des Satzes versorgt werden, der hinzugefügt, überschrieben oder gelöscht werden soll. Wird bei WRITE die Nummer eines bereits vorhandenen Satzes bzw. bei REWRITE bzw. DELETE die Nummer eines nicht existierenden Satzes (z.B. eines Leersatzes)angegeben, tritt eine INVALID KEY-Bedingung auf und WRITE, REWRITE oder DELETE verzweigen zur INVALID KEY-Anweisung bzw. zur vereinbarten USE-Prozedur.

– ACCESS MODE IS DYNAMIC

  gestattet es, die Datei sowohl sequentiell als auch wahlfrei zu verarbeiten. Die jeweilige Zugriffsart wird dabei über das Format der READ-Anweisung gewählt.

9.3.5 Erstellen einer relativen Datei mit wahlfreiem Zugriff

Das folgende Beispiel gibt ein einfaches COBOL-Programm wieder, mit dem eine relative Datei mit wahlfreiem Zugriff erstellt werden kann. Die Datensätze können dabei in beliebiger Reihenfolge in die Datei geschrieben werden.

Beispiel 9-9: Programm zum wahlfreien Erstellen einer relativen Datei

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  RELATIV.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT RELATIV-DATEI
    ASSIGN TO "RELFILE"
    ORGANIZATION IS RELATIVE
    ACCESS MODE IS RANDOM ------------------------------------------------ (1)
    RELATIVE KEY IS REL-KEY ---------------------------------------------- (2)
    FILE STATUS IS FS-CODE DVS-CODE. ------------------------------------- (3)
DATA DIVISION.
FILE SECTION.
FD  RELATIV-DATEI.
01  RELATIV-SATZ                PIC X(33).
WORKING-STORAGE SECTION.
01  REL-KEY                     PIC 9(3).
    88  EINGABE-ENDE                        VALUE ZERO.
01  EIN-AUSGABE-ZUSTAND.
    05  FS-CODE                 PIC 9(2).
    05  DVS-CODE.
        06  DVS-CODE-1          PIC 9(2) COMP.
            88  DVS-CODE-2-DEFINIERT        VALUE 64.
        06  DVS-CODE-2          PIC X(4).
01  CLOSE-SCHALTER              PIC X       VALUE "0".
    88  DATEI-OFFEN                         VALUE "1".
    88  DATEI-GESCHLOSSEN                   VALUE "0".
01  RELATIV-TEXT.
    05                          PIC X(24)
      VALUE "******DIES IST SATZ NR. ".
    05  SATZNR                  PIC 9(3).
    05                          PIC X(6)    VALUE "$$$$$$".
PROCEDURE DIVISION.
DECLARATIVES.
AUSGABE-FEHLER SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON RELATIV-DATEI.
PERMANENTER-FEHLER. -------------------------------------------------------- (4)
    IF FS-CODE NOT LESS THAN 30
       DISPLAY "NICHT BEHEBBARER FEHLER FUER RELATIV-DATEI"
          UPON T
       DISPLAY "FILE STATUS: " FS-CODE UPON T
       IF DVS-CODE-2-DEFINIERT
          DISPLAY "DVS-CODE:    " DVS-CODE-2 UPON T
       END-IF
       IF DATEI-OFFEN
```

```
           CLOSE RELATIV-DATEI
        END-IF
        DISPLAY "PROGRAMM ABNORMAL BEENDET" UPON T
        STOP RUN
     END-IF.
 AUSGABE-FEHLER-ENDE.
        EXIT.
 END DECLARATIVES.
 VORLAUF.
     OPEN OUTPUT RELATIV-DATEI
     SET DATEI-OFFEN TO TRUE.
 DATEI-LADEN.
     PERFORM SATZNUMMER-EINLESEN
        WITH TEST AFTER
        UNTIL REL-KEY IS NUMERIC
     PERFORM WITH TEST BEFORE UNTIL EINGABE-ENDE
        WRITE RELATIV-SATZ FROM RELATIV-TEXT
           INVALID KEY ------------------------------------------------- (5)
              DISPLAY "SATZ MIT NR. " REL-KEY
                 "IST BEREITS IN DER DATEI" UPON T
        END-WRITE
        PERFORM SATZNUMMER-EINLESEN
           WITH TEST AFTER
           UNTIL REL-KEY IS NUMERIC
     END-PERFORM.
 NACHLAUF.
     SET DATEI-GESCHLOSSEN TO TRUE
     CLOSE RELATIV-DATEI
     STOP RUN.
 SATZNUMMER-EINLESEN.
     DISPLAY "BITTE SATZNUMMER EINGEBEN; DREISTELLIG MIT FUEHRENDE
-       "N NULLEN" UPON T
     DISPLAY "PROGRAMM BEENDEN MIT '000'" UPON T
     ACCEPT REL-KEY FROM T
     IF REL-KEY NUMERIC
        THEN MOVE REL-KEY TO SATZNR
        ELSE DISPLAY "EINGABE MUSS NUMERISCH SEIN" UPON T
     END-IF.
```

(1) Die ACCESS MODE-Klausel vereinbart wahlfreien Zugriff auf die Sätze der Datei RELATIV-DATEI. Sie können also bei der Erstellung in beliebiger Reihenfolge in die Datei geschrieben werden.

(2) Die RELATIVE KEY-Angabe legt REL-KEY als Schlüsselfeld für die relativen Satznummern fest. Es wird in der WORKING-STORAGE SECTION als dreistelliges numerisches Datenfeld vereinbart,

(3) In der FILE STATUS-Klausel wird von der Möglichkeit Gebrauch gemacht, dem Programm zusätzlich zum FILE STATUS-Code auch den Fehlercode des DVS zur Verfügung zu stellen. Die Datenfelder zur Aufnahme dieser Informationen werden in der WORKING-STORAGE SECTION vereinbart und in den DECLARATIVES ausgewertet.

(4) Die DECLARATIVES sehen lediglich eine Prozedur für nicht behebbare Ein-/Ausgabefehler (FILE STATUS ≥ 30) vor, da eine Endebedingung bei Ausgabedateien nicht auftreten kann und Schlüsselfehler über INVALID KEY abgefangen werden.

(5) Eine INVALID KEY-Bedingung bei wahlfreiem WRITE tritt auf, wenn der Satz mit der zugehörigen relativen Satznummer bereits in der Datei vorhanden ist.

9.3.6 Ein-/Ausgabezustände

Jeder Datei im Programm können mit der FILE STATUS-Klausel Datenfelder zugeordnet werden, in denen das Laufzeitsystem nach jedem Zugriff auf die Datei Informationen darüber hinterlegt,

– ob die Ein-/Ausgabeoperation erfolgreich war und

– welcher Art ggf. die dabei aufgetretenen Fehler sind.

Diese Informationen können z.B. in den DECLARATIVES durch USE-Prozeduren ausgewertet werden und gestatten eine Analyse von Ein-/Ausgabefehlern durch das Programm. Als Erweiterung zum COBOL-Standard bietet COBOL85 die Möglichkeit, in diese Analyse auch die Schlüssel der DVS-Fehlermeldungen einzubeziehen. Dadurch läßt sich eine feinere Differenzierung der Fehlerursachen erreichen. Die FILE STATUS-Klausel wird im FILE-CONTROL-Paragraphen der ENVIRONMENT DIVISION angegeben; ihr Format ist (siehe [1]):

```
FILE STATUS IS datenname-1 [datenname-2]
```

Dabei müssen datenname-1 und, falls angegeben, datenname-2 in der WORKING-STORAGE SECTION oder der LINKAGE SECTION definiert sein. Für die Formate und die möglichen Werte dieser beiden Datenfelder gelten folgende Regeln:

datenname-1

– muß als zwei Byte langes numerisches oder alphanumerisches Datenfeld erklärt werden, also z.B.

```
01 datenname-1          PIC X(2).
```

– enthält nach jeder Ein-/Ausgabeoperation auf die zugeordnete Datei einen zweistelligen numerischen Zustandscode, dessen Bedeutung der Tabelle am Ende dieses Abschnitts entnommen werden kann.

datenname-2

– muß als sechs Byte langes Gruppenfeld der folgenden Struktur erklärt werden:

```
01 datenname-2.
   02 datenname-2-1       PIC 9(2) COMP.
   02 datenname-2-2       PIC X(4).
```

– dient der Aufnahme des DVS-Fehlerschlüssels (DVS-Codes) zum jeweiligen Ein-/Ausgabezustand und enthält nach jedem Zugriff auf die zugeordnete Datei einen Wert, der vom Inhalt des Feldes datenname-1 abhängt und sich aus folgender Zusammenstellung ergibt:

Inhalt von datenname-1 ungleich 0?	DVS-Code ungleich 0?	Wert von datenname-2-1	Wert von datenname-2-2
nein	nicht relevant	undefiniert	undefiniert
ja	nein	0	undefiniert
ja	ja	64	DVS-Code der zugeordneten Fehlermeldung

Die DVS-Codes und die zugeordneten Fehlermeldungen können Handbuch [4] entnommen werden.

Tabelle 9-14: Ein-/Ausgabezustände für relative Dateien

Ein-/Ausgabe-Zustand	Bedeutung
	Erfolgreiche Ausführung
00	Die Ein-/Ausgabe-Anweisung wurde erfolgreich ausgeführt. Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar.
04	Satzlängenkonflikt: Eine READ-Anweisung wurde erfolgreich ausgeführt. Die Länge des gelesenen Datensatzesliegt jedoch nicht in den Grenzen, die durch die Satzbeschreibungen der Datei festgelegt wurden.
05	Erfolgreicher OPEN INPUT/I-O/EXTEND auf eine Datei mit OPTIONAL-Angabe in der SELECT-Klausel, die zum Zeitpunkt der Ausführung der OPEN-Anweisung nicht vorhanden war
	Erfolglose Ausführung: Endebedingung
10	Es wurde versucht, eine READ-Anweisung auszuführen. Es war jedoch kein nächster logischer Datensatz vorhanden, da das Dateiende erreicht war (sequentielles READ). Es wurde zum ersten Mal versucht, eine READ-Anweisung für eine nicht vorhandene Datei mit OPTIONAL-Angabe auszuführen.
14	Es wurde versucht, eine READ-Anweisung auszuführen. Das durch RELATIVE KEY beschriebene Datenfeld ist aber zu klein, um die relative Satznummer aufzunehmen (sequentielles READ).
	Erfolglose Ausführung: Schlüsselfehlerbedingung
22	Doppelter Schlüssel Es wurde versucht, eine WRITE-Anweisung mit einem Schlüssel auszuführen, für den in der Datei bereits ein Satz vorhanden ist.
23	Datensatz nicht gefunden oder Satzschlüssel Null Es wurde versucht, anhand eines Schlüssels mit einer READ-, START-, DELETE- oder REWRITE-Anweisung auf einen Datensatz zuzugreifen, der in der Datei nicht vorhanden ist, oder der Zugriff erfolgte mit Satzschlüssel Null
24	Überschreiten der Bereichsgrenzen Es wurde versucht, eine WRITE-Anweisung außerhalb der vom System festgelegten Bereichsgrenzen einer relativen Datei auszuführen (unzureichende Sekundärzuweisung im FILE-Kommando) oder eine WRITE-Anweisung im sequentiellen Zugriffsmodus zu geben, bei der die relative Satznummer so groß ist, daß sie nicht in das mit der RELATIVE KEY-Angabe beschriebene Datenfeld paßt.

Ein-/Ausgabe-Zustand	Bedeutung
	Erfolglose Ausführung: Permanenter Fehler
30	Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar.
35	Es wurde versucht, eine OPEN INPUT/I-O-Anweisung für eine Datei auszuführen, die nicht vorhanden ist.
37	OPEN-Anweisung auf eine Datei, die aufgrund folgender Bedingungen nicht eröffnet werden kann: 1. OPEN OUTPUT/I-O/EXTEND auf eine schreibgeschützte Datei (Paßwort, RETENTION-PERIOD, ACCESS=READ) 2. OPEN INPUT auf eine lesegeschützte Datei (Paßwort)
38	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die vorher mit der LOCK-Angabe geschlossen wurde.
39	Die OPEN-Anweisung war aus einem der folgenden Gründe erfolglos: 1. Im SET-FILE-LINK-Kommando wurde einer oder mehrere der Operanden ACCESS-METHOD, RECORD-FORMAT bzw. RECORD-SIZE mit Werten angegeben, die von den entsprechenden expliziten oder impliziten Programmangaben abweichen. 2. Für eine Eingabedatei stimmt der Katalogeintrag des Operanden FCBTYPE nicht mit der entsprechenden expliziten oder impliziten Programmangab bzw. mit der entsprechenden Angabe im SET-FILE-LINK-Kommando überein. 3. Für eine Datei, die mit der DVS-Zugriffsmethode UPAM verarbeitet werden soll, wurde variable Satzlänge vereinbart.
	Erfolglose Ausführung: Logischer Fehler
41	Es wurde versucht, eine OPEN-Anweisung für eine Dateiauszuführen, die bereits eröffnet ist.
42	Es wurde versucht, eine CLOSE-Anweisung für eine Datei auszuführen, die nicht eröffnet ist.
43	Bei ACCESS MODE IS SEQUENTIAL: Die letzte vor Ausführung einer DELETE- oder REWRITE-Anweisung ausgeführte Ein-/Ausgabe-Anweisungwar keine erfolgreich ausgeführte READ-Anweisung.
44	Überschreiten der Satzlängengrenzen: Es wurde versucht, eine WRITE- oder REWRITE-Anweisung auszuführen. Die Länge des Datensatzes liegt jedoch nicht in dem für diese Datei zulässigen Bereich.

Ein-/Ausgabe-Zustand	Bedeutung
46	Es wurde versucht, eine sequentielle READ-Anweisung für eine Datei auszuführen, die sich im Eröffnungsmodus INPUT oder I-O befindet; ein nächster gültiger Datensatz steht aber nicht zur Verfügung. Grund: 1. Die vorhergehende START-Anweisung war erfolglos, oder 2. die vorhergehende READ-Anweisung war erfolglos, ohne eine Endebedingung zu verursachen, oder 3. die vorhergehende READ-Anweisung hat eine Ende-Bedingung verursacht.
47	Es wurde versucht, eine READ- oder START-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus INPUT oder I-O befindet.
48	Es wurde versucht, eine WRITE-Anweisung für eine Datei auszuführen, die sich – bei sequentiellem Zugriff nicht im Eröffnungsmodus OUTPUT oder EXTEND, – bei wahlfreiem oder dynamischem Zugriff nicht im Eröffnungsmodus OUTPUT oder I-O befindet.
49	Es wurde versucht, eine DELETE- oder REWRITE-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus I-O befindet.
	Sonstige erfolglose Ausführungen
90	Systemfehler; es ist keine weitere Information über die Ursache vorhanden.
91	Systemfehler; OPEN-Fehler
93	Nur bei Simultanverarbeitung (siehe Abschnitt 9.5): Die Ein-/Ausgabe-Anweisung konnte nicht erfolgreich durchgeführt werden, weil ein anderer Prozeß auf dieselbe Datei zugreift und die Zugriffe nicht vereinbar sind.
94	Nur bei Simultanverarbeitung (siehe Abschnitt 9.5): die Aufruffolge READ - REWRITE/DELETE wurde nicht eingehalten.
95	Unverträglichkeit zwischen den Angaben im BLOCK-CONTROL-INFO- oder BUFFER-LENGTH-Operanden des SET-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder demFormat des verwendeten Datenträgers

9.4 Indizierte Dateiorganisation

9.4.1 Merkmale indizierter Dateiorganisation

In einer indiziert organisierten Datei enthält jeder Datensatz einen Schlüssel, d.h. eine Folge beliebiger (auch nichtabdruckbarer) Zeichen, die ihn (innerhalb der Datei) eindeutig identifizieren. Die Anfangspositionen (KEYPOS) und Längen (KEYLEN) der Schlüssel stimmen dabei für alle Sätze einer Datei überein.

Mit Hilfe eines im Programm vereinbarten Schlüsselfeldes, das Lage und Länge des Schlüssels im Datensatz beschreibt, kann über diesen Satzschlüssel direkt (wahlfrei) auf jeden Satz der Datei zugegriffen werden. Zusätzlich zu den Möglichkeiten der sequentiellen Dateiorganisation gestattet dies, in einer indizierten Datei

– Sätze wahlfrei zu erstellen

– Sätze wahlfrei zu lesen und zu aktualisieren,

– nachträglich Sätze einzufügen und

– bereits vorhandene Datensätze logisch zu löschen.

Für die Bearbeitung indizierter Dateien verwenden COBOL-Programme die Zugriffsmethode ISAM des DVS (siehe [4]). Sie gestattet es mehreren Anwendern, gleichzeitig eine Datei zu aktualisieren (siehe 9.5).

Indizierte Dateien können ausschließlich auf Plattenspeichern eingerichtet werden.

Dateistruktur

Eine ausführliche Beschreibung des Aufbaus einer ISAM-Datei findet sich in [4]; die folgende Darstellung ist lediglich eine kurze Zusammenfassung der wichtigsten Tatsachen:

Eine ISAM-Datei besteht aus zwei Komponenten mit unterschiedlichen Funktionen,

– den Indexblöcken und

– den Datenblöcken

Falls private Datenträger verwendet werden, können Index- und Datenblöcke auf verschiedenen Datenträgern liegen.

– Die Datenblöcke enthalten die Datensätze des Anwenders. Diese sind in aufsteigender Reihenfolge ihrer Schlüssel logisch miteinander verkettet; ihre physische Reihenfolge auf dem Datenträger ist beliebig.

  Datenblöcke können eine Länge von einem PAM-Block (2048 byte) oder einem ganzzahligen Vielfachen davon (bis zu 16 PAM-Blöcken) haben.

– Die Indexblöcke dienen dem Auffinden der Datensätze über die Satzschlüssel. Sie lassen sich verschiedenen Indexstufen zuordnen:

  Indexblöcke der niedrigsten Stufe enthalten Zeiger auf Datenblöcke, die Indexblöcke höherer Stufe Zeiger auf die Indexblöcke der nächstniedrigeren Stufe.

  Der Indexblock der höchsten Stufe wird immer in der Datei angelegt, auch wenn sie keine Datensätze enthält. Neben den Zeigern enthält er eine 36 byte lange ISAM-Etikettinformation.

  Die Einträge in den Indexblöcken sind physisch stets in der Reihenfolge aufsteigender Satzschlüssel angeordnet; sie müssen daher reorganisiert werden, wenn in der darunterliegenden Stufe neue Index- bzw. Datenblöcke entstehen.

  Indexblöcke haben eine feste Länge von einem PAM-Block.

Blockteilung

Beim Erweitern einer ISAM-Datei wird jeder neue Datensatz in den Datenblock eingefügt, zu dem er auf Grund seines Satzschlüssels gehört.
Dabei kann es vorkommen, daß in diesem Block kein Platz zur Aufnahme eines weiteren Satzes zur Verfügung steht. In diesem Fall kommt es zu Blockteilung: Der alte Datenblock wird geteilt, die entstandenen Hälften werden in neue (leere) Blöcke übertragen. Der alte Datenblock bleibt der Datei zugeordnet und wird als freier Datenblock gekennzeichnet (siehe DVS-Benutzerhandbuch [4]).

Häufige Blockteilungen verlangsamen die Verarbeitung. Ihre Zahl kann aber vermindert werden, wenn bereits bei der Dateierstellung in den Datenblöcken Platz für künftige Erweiterungen reserviert wird: Bei der Zuweisung der Ausgabedatei kann man durch die Angabe des Operanden PADDING-FACTOR im ACCESS-METHOD=ISAM-Parameter des SET-FILE-LINK-Kommandos ereichen, daß der darin vereinbarte Prozentsatz eines Datenblokkes beim Laden der Datei für spätere Erweiterung freibleibt.

Beispiel 9-10: PADDING-FACTOR-Operand bei der Zuweisung einer ISAM-Datei

Beim Neuerstellen der Datei ISAM.AUSGABE steht nur etwa jeder vierte Datensatz zur Verfügung. 75% eines jeden Datenblockes sollen daher für künftige Erweiterungen reserviert werden. Dies wird über das folgende SET-FILE-LINK-Kommando vereinbart:

```
/SET-FILE-LINK AUSDAT,ISAM.AUSGABE,ACCESS-METHOD=ISAM(PADDING-FACTOR=75)
```

9.4.2 COBOL-Sprachmittel für die Verarbeitung indizierter Dateien

Das folgende Programmskelett gibt einen Überblick über die wichtigsten Klauseln und Anweisungen, die COBOL85 für die Verarbeitung indizierter Dateien zur Verfügung stellt. Die wesentlichen Angaben werden im Anschluß daran kurz erläutert:

```
IDENTIFICATION DIVISION.
...
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT interner-dateiname
    ASSIGN TO externer-name
    ORGANIZATION IS INDEXED
    ACCESS MODE IS zugriffsart
    RECORD KEY IS primärschlüssel
    ALTERNATE RECORD KEY IS sekundärschlüssel
    FILE STATUS IS statusfelder.
    ...
DATA DIVISION.
FILE SECTION.
FD  interner-dateiname.
    BLOCK CONTAINS blocklängenangabe
    RECORD satzlängenangabe
    ...
01  datensatz.
    nn feld-1  typ&länge
    ...
    nn primärschlüssel-feld  typ&länge
    nn sekundärschlüssel-feld  typ&länge
    ...
PROCEDURE DIVISION.
    ...
    OPEN open-modus interner-dateiname
    ...
    START interner-dateiname
    ...
    READ interner-dateiname
    ...
    REWRITE datensatz
    ...
    WRITE datensatz
    ...
    DELETE interner-dateiname
    ...
    CLOSE interner-dateiname
    ...
    STOP RUN.
```

`SELECT interner-dateiname`

legt den Namen fest, unter dem die Datei im Quellprogramm angesprochen wird.

interner-dateiname muß ein gültiges Programmiererwort sein.

Das Format der SELECT-Klausel erlaubt auch die Angabe OPTIONAL für Eingabedateien, die beim Programmablauf nicht unbedingt vorhanden sein müssen.

Ist einem mit SELECT OPTIONAL vereinbarten Dateinamen beim Programmablauf keine Datei zugewiesen, so wird

– bei OPEN INPUT im Dialogbetrieb der Programmablauf mit der Meldung COB9117 unterbrochen und ein SET-FILE-LINK-Kommando angefordert, im Stapelbetrieb die AT END-Bedingung ausgelöst,
– bei OPEN I-O oder OPEN EXTEND eine Datei mit dem Namen FILE.COB85.linkname angelegt.

`ASSIGN TO externer-name`

gibt die Systemdatei an, die der Datei zugewiesen wird, oder legt den Linknamen fest, über den eine katalogisierte Datei zugeordnet werden kann.

externer-name muß entweder
– ein zulässiges Literal,
– ein in der DATA DIVISION definierter zulässiger Datenname oder
– ein gültiger Herstellername

aus dem Format der ASSIGN-Klausel sein (siehe [1]).

`ORGANIZATION IS INDEXED`

legt fest, daß die Datei indiziert organisiert ist.

`ACCESS MODE IS zugriffsart`

bestimmt die Art, in der auf die Sätze zugegriffen werden kann.
Für zugriffsart sind folgende Angaben möglich (siehe auch 9.4.4):

SEQUENTIAL	legt fest, daß die Sätze nur sequentiell verarbeitet werden.
RANDOM	vereinbart, daß auf die Sätze nur wahlfrei zugegriffen wird.
DYNAMIC	gestattet, daß auf die Sätze wahlweise sequentiell oder wahlfrei zugegriffen wird.

Die ACCESS MODE-Klausel ist optional. Wird sie nicht angegeben, nimmt der Compiler ACCESS MODE IS SEQUENTIAL an.

`RECORD KEY IS primärschlüssel`

gibt das Feld innerhalb des Datensatzes an, das den primären Satzschlüssel enthält.

primärschlüssel muß als Datenfeld innerhalb der zugehörigen Datensatzerklärung vereinbart werden (siehe unten).

Außer beim sequentiellen Lesen muß primärschlüssel vor jeder Ein-/Ausgabeanweisung mit dem Primärschlüssel des Satzes versorgt werden, der bearbeitet werden soll.

`ALTERNATE RECORD KEY IS sekundärschlüssel`

Mit COBOL-Programmen können auch Dateien verarbeitet werden, deren Datensätze außer dem obligatorischen Primärschlüssel (RECORD KEY)
einen oder mehrere Sekundärschlüssel (ALTERNATE RECORD KEY) enthalten.
Sind in einer Datei Sekundärschlüssel definiert, kann der Benutzer auf die Datensätze entweder über den Primärschlüssel oder über den/die Sekundärschlüssel zugreifen.

sekundärschlüssel muß als Datenfeld innerhalb der zugehörigen Datensatzerklärung vereinbart werden (siehe unten).

`FILE STATUS IS statusfelder`

gibt die Datenfelder an, in denen das Laufzeitsystem nach jedem Zugriff auf die Datei Informationen darüber hinterlegt,

– ob die Ein-/Ausgabeoperation erfolgreich war und

– welcher Art ggf. die dabei aufgetretenen Fehler sind.

Die statusfelder müssen in der WORKING-STORAGE oder der LINKAGE SECTION vereinbart werden. Ihr Format und die Bedeutung der einzelnen Zustandscodes werden in Abschnitt 9.4.6 beschrieben.

Die FILE STATUS-Klausel ist optional. Wird sie nicht angegeben, stehen dem Programm die oben erwähnten Informationen nicht zur Verfügung.

`BLOCK CONTAINS blocklängenangabe`

legt die maximale Größe eines logischen Blocks fest. Sie bestimmt, wie viele Datensätze jeweils gemeinsam durch eine Ein-/Ausgabeoperation in den bzw. aus dem Puffer des Programms übertragen werden sollen.

blocklängenangabe muß dabei eine zulässige Angabe aus dem Format der BLOCK CONTAINS-Klausel sein.

Die Blockung von Datensätzen verringert

– die Zahl der Zugriffe auf periphere Speicher und damit die Laufzeit des Programms und

– die Zahl der Blockzwischenräume auf dem Speichermedium und damit den physischen Platzbedarf der Datei.

Andererseits wird bei Zugriffen mit Sperrmechanismus im Verlauf einer Simultanverarbeitung (siehe 9.5) stets der gesamte Block gesperrt, in dem sich der aktuelle Satz befindet. Ein großer Blockungsfaktor führt in diesem Fall daher zu Einbußen an Verarbeitungsgeschwindigkeit.

Der Compiler errechnet bei der Übersetzung aus den Angaben im Quellprogramm über Block-und Satzlänge einen Wert für die Puffergröße, der vom Laufzeitsystem für das DVS auf das nächstgrößere Vielfache eines PAM-Blocks (2048 byte) aufgerundet wird. Diese Voreinstellung kann bei der Dateizuweisung durch die Angabe des BLKSIZE-Operanden im SET-FILE-LINK-Kommando verändert werden (siehe 9.1.3), wobei darauf zu achten ist, daß

– der Puffer mindestens so groß sein muß wie der längste Datensatz und

– bei Verarbeitung im keylosen Format (BLKCTRL = DATA) die Verwaltungsinformationen ("Pamkey") im Puffer Platz finden (siehe 9.1.4).

Außer bei neu angelegten Dateien (OPEN OUTPUT) hat die im Katalog eingetragene Blockgröße stets Vorrang gegenüber den Blockgrößenangaben im Programm bzw. im SET-FILE-LINK-Kommando.

Die BLOCK CONTAINS-Klausel ist optional. Wird sie nicht angegeben, nimmt der Compiler BLOCK CONTAINS 1 RECORD an, d.h. die Datensätze werden nicht geblockt.

`RECORD satzlängenangabe`

– legt fest, ob Sätze fester oder variabler Länge verarbeitet werden sollen und

– bestimmt bei Sätzen variabler Länge einen Bereich für die zulässigen Satzgrößen und, falls im Format angegeben, ein Datenfeld zur Aufnahme der jeweils aktuellen Satzlängeninformation.

Die satzlängenangabe muß einem der drei Formate der RECORD-Klausel entsprechen, die COBOL85 zur Verfügung stellt. Sie darf nicht im Widerspruch zu den Satzlängen stehen, die der Compiler aus den Angaben der dazugehörigen Datensatzerklärung(en) errechnet.

Die RECORD-Klausel ist optional. Wird sie nicht angegeben, nimmt der Compiler Sätze variabler Länge an.

```
01 datensatz.
   nn feld-1           typ&länge
   ...
   nn primärschlüssel        typ&länge
   ...
   nn sekundärschlüssel        typ&länge
```

stellt eine Datensatzerklärung für die zugehörige Datei dar. Sie beschreibt den logischen Aufbau von Datensätzen.

Für jede Datei ist mindestens eine Datensatzerklärung erforderlich. Werden für eine Datei mehrere Datensatzerklärungen angegeben, ist das vereinbarte Satzformat zu beachten:
– Bei Sätzen fester Länge müssen alle Satzerklärungen die gleiche Größe haben,
– bei Sätzen variabler Länge dürfen sie nicht im Widerspruch zur Satzlängenangabe der RECORD-Klausel stehen. Darüberhinaus muß auch die Datensatzerklärung mit der kleinsten Satzlänge den Satzschlüssel noch ganz enthalten.

Mindestens eine der Datensatzerklärungen muß das Primärschlüsselfeld explizit als Teilfeld von datensatz vereinbaren. Für typ&länge sind die erforderlichen Längen- und Formatvereinbarungen (PICTURE- und USAGE-Klauseln etc.) einzusetzen (primärschlüssel darf bis zu 255 byte lang sein).

sekundärschlüssel ist der Datenname aus der entsprechenden ALTERNATE RECORD KEY-Klausel. Jedes Sekundärschlüssel-Feld darf maximal 127 Byte lang sein. Überlappungen mit dem Primärschlüssel oder weiteren Sekundärschlüsseln sind zulässig, sofern zwei Schlüsselfelder nicht an derselben Stelle beginnen. Der COBOL85-Compiler läßt auch rein numerisch (PIC 9) oder alphabetisch (PIC A) definierte Sekundärschlüssel zu.

Für alle anderen Datensatzerklärungen zu dieser Datei ist die Unterteilung von datensatz in Teilfelder (feld-1, feld-2,...) optional.

```
OPEN open-modus interner-dateiname
```

eröffnet die Datei in der angegebenen Eröffnungsart open-modus für die Verarbeitung.

Für open-modus sind folgende Angaben möglich:

INPUT	eröffnet die Datei als Eingabedatei; sie kann nur gelesen werden
OUTPUT	eröffnet die Datei als Ausgabedatei; sie kann nur neu geschrieben werden.
EXTEND	eröffnet die Datei als Ausgabedatei; sie kann erweitert werden.
I-O	eröffnet die Datei als Ein-/Ausgabedatei; sie kann (Satz für Satz) gelesen, aktualisiert und zurückgeschrieben werden.

Die Angabe für open-modus legt fest, mit welchen Ein-/Ausgabeanweisungen auf die Datei zugegriffen werden darf (siehe 9.4.4).

```
START interner-dateiname
READ interner-dateiname
REWRITE datensatz
WRITE datensatz
DELETE interner-dateiname
```

sind Ein-/Ausgabeanweisungen für die Datei, die jeweils

– in der Datei auf einen Satz positionieren bzw.

– einen Satz lesen bzw.

– einen Satz zurückschreiben bzw.

– einen Satz schreiben bzw.

– einen Satz löschen

Welche dieser Anweisungen für die Datei zulässig sind, hängt von der Eröffnungsart ab, die in der OPEN-Anweisung vereinbart wird. Dieser Zusammenhang wird in Abschnitt 9.4.4 beschrieben.

```
CLOSE interner-dateiname
```

beendet die Verarbeitung der Datei.

Durch die zusätzliche Angabe WITH LOCK kann ein erneutes Eröffnen der Datei im selben Programmlauf verhindert werden.

9.4.3 Zulässige Satzformate und Zugriffsarten

Satzformate

Indizierte Dateien können Sätze fester Länge (RECFORM=F) oder variabler Länge (RECFORM=V) enthalten. In beiden Fällen können die Sätze geblockt oder ungeblockt vorliegen.
Im COBOL-Quellprogramm kann das Format der zu verarbeitenden Sätze in der RECORD-Klausel festgelegt werden. Welche Angaben dabei dem jeweiligen Satzformat zugeordnet sind, ist in der folgenden Tabelle zusammengestellt:

Satzformat	Angabe in der RECORD-Klausel
Sätze fester Länge	RECORD CONTAINS...CHARACTERS (Format 1)
Sätze variabler Länge	RECORD IS VARYING IN SIZE... (Format 2) oder RECORD CONTAINS...TO... (Format 3)

Tabelle 9-15: Festlegen von Satzformaten in der RECORD-Klausel

Zugriffsarten

Auf Sätze einer indizierten Datei kann sequentiell, wahlfrei oder dynamisch zugegriffen werden.
Im COBOL-Quellprogramm wird die Zugriffsart durch die ACCESS MODE-Klausel festgelegt. Die folgende Übersicht stellt die möglichen Angaben und ihre Auswirkungen auf die Zugriffsart zusammen:

ACCESS MODE-Klausel	**Zugriffsart**
SEQUENTIAL	Sequentieller Zugriff: Die Datensätze können nur in aufsteigender Reihenfolge ihrer Satzschlüssel verarbeitet werden. Das bedeutet: – Beim Lesen wird jeweils der nächste logische Datensatz zur Verfügung gestellt. – Beim Schreiben wird jeweils der nächste logische Datensatz in die Datei ausgegeben.
RANDOM	Wahlfreier Zugriff: Die Datensätze können in beliebiger Reihenfolge über ihre Satzschlüssel angesprochen werden. Dazu muß vor jeder Ein-/Ausgabeanweisung für einen Satz dessen Schlüssel (Primär- oder Sekundärschlüssel) im (ALTERNATE) RECORD KEY-Feld zur Verfügung gestellt werden.
DYNAMIC	Dynamischer Zugriff: Auf die Datensätze kann sowohl sequentiell als auch wahlfrei zugegriffen werden. Die jeweilige Zugriffsart wird dabei über das Format der Ein-/Ausgabeanweisung gewählt.

Tabelle 9-16: ACCESS MODE-Klausel und Zugriffsart

9.4.4 Eröffnungsarten und Verarbeitungsformen

Mit den Sprachmitteln eines COBOL-Programms lassen sich indizierte Dateien
– erstellen,
– lesen,
– durch Hinzufügen neuer Datensätze erweitern und
– durch Abändern oder Löschen vorhandener Datensätze aktualisieren.

Welche Ein-/Ausgabeanweisungen im Programm jeweils für eine Datei zulässig sind wird dabei durch ihren Eröffnungsmodus bestimmt, der in der OPEN-Anweisung angegeben wird:

`OPEN OUTPUT`

Als Ein-/Ausgabeanweisung ist unabhängig von der Angabe in der ACCESS MODE-Klausel WRITE mit folgendem Format erlaubt:

```
WRITE...[FROM...]
        [INVALID KEY...]
        [NOT INVALID KEY...]
        [END-WRITE]
```

In diesem Modus können indizierte Dateien ausschließlich neu erstellt (geladen) werden.

– ACCESS MODE IS SEQUENTIAL

  erlaubt es, eine indizierte Datei sequentiell zu erstellen. Dabei müssen die Datensätze der WRITE-Anweisung aufsteigend nach ihren Satzschlüsseln sortiert zur Verfügung gestellt werden.
  Das RECORD KEY-Feld muß vor jeder WRITE-Anweisung mit dem Satzschlüssel des auszugebenden Datensatzes versorgt werden. Dabei muß jeder neue Schlüssel grösser sein als der zuletzt angegebene. Ist dies nicht der Fall, tritt eine INVALID KEY-Bedingung auf und WRITE verzweigt zur INVALID KEY-Anweisung bzw. zur vereinbarten USE-Prozedur, ohne den Satz zu schreiben. Ein Überschreiben von Datensätzen ist hier also nicht möglich.

– ACCESS MODE IS DYNAMIC oder RANDOM

  erlaubt es, eine indizierte Datei wahlfrei zu erstellen. Dabei ist zu beachten, daß das Erstellen nach aufsteigenden Satzschlüsseln effizienter abläuft.

`OPEN EXTEND`

Mit OPEN EXTEND kann eine vorhandene Datei erweitert werden. Die ACCESS MODE-Klausel wird wie bei OPEN OUTPUT verwendet.

`OPEN INPUT`

Welche Ein-/Ausgabeanweisungen bzw. Anweisungsformate erlaubt sind, hängt von der Angabe in der ACCESS MODE-Klausel ab. Die folgende Tabelle stellt die Möglichkeiten für OPEN INPUT zusammen:

Anweisung	**Eintrag in der ACCESS MODE-Klausel**		
	SEQUENTIAL	RANDOM	DYNAMIC
START	START... [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-START]	Anweisung nicht zulässig	START... [KEY IS...] [INVALID KEY...] [NOT INVALID KEY] [END-START]
READ	READ...[NEXT] [INTO...] [AT END...] [NOT AT END...] [END-READ...]	READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]	Für sequentiellen Zugriff: READ...NEXT [INTO...] [AT END...] [NOT AT END...] [END-READ]
			Für wahlfreien Zugriff: READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]

Tabelle 9-17: Erlaubte Ein-/Ausgabeanweisungen für OPEN INPUT

In diesem Modus können indizierte Dateien gelesen werden. Abhängig von der vereinbarten Zugriffsart hat die READ-Anweisung dabei folgende Wirkung:

– ACCESS MODE IS SEQUENTIAL

  erlaubt es ausschließlich, die Datei sequentiell zu lesen. READ stellt dabei die Datensätze in der Reihenfolge aufsteigender Satzschlüssel zur Verfügung.
  Das (ALTERNATE) KEY-Schlüsselfeld wird von READ nicht ausgewertet. Vor der Ausführung einer READ-Anweisung kann jedoch mit Hilfe von START auf einen beliebigen Satz der Datei positioniert werden: Über eine Vergleichsbedingung legt START den Schlüssel des zuerst zu lesenden Satzes und damit den Ausgangspunkt für nachfolgende sequentielle Leseoperationen fest (siehe auch 9.4.5). Kann die Vergleichsbedingung von keinem Satzschlüssel der Datei erfüllt werden, tritt eine INVALID KEY-Bedingung auf und START verzweigt zur INVALID KEY-Anweisung bzw. zur vereinbarten USE-Prozedur.

– ACCESS MODE IS RANDOM

  ermöglicht es, die Sätze der Datei wahlfrei zu lesen. READ stellt dabei die Datensätze in beliebiger Reihenfolge zur Verfügung; der Zugriff auf jeden Satz erfolgt über seinen Satzschlüssel Das (ALTERNATE) KEY-Feld muß dazu vor jeder READ-Anweisung mit dem Schlüssel des Satzes versorgt werden, der gelesen werden soll.
  Wird der Schlüssel eines nicht existierenden Satzes angegeben, tritt eine INVALID KEY-Bedingung auf und READ verzweigt zur INVALID KEY-Anweisung bzw. zur vereinbarten USE-Prozedur.

– ACCESS MODE IS DYNAMIC

  gestattet es, die Datei sowohl sequentiell als auch wahlfrei zu lesen.
  Die jeweilige Zugriffsart wird dabei über das Format der READ-Anweisung gewählt (siehe Tabelle 9-18).
  Eine START-Anweisung ist nur für sequentielles Lesen sinnvoll.

```
OPEN I-O
```

Welche Ein-/Ausgabeanweisungen bzw. Anweisungsformate erlaubt sind, hängt von der Angabe in der ACCESS MODE-Klausel ab.

In diesem Modus können in einer indizierten Datei Sätze
– gelesen,
– hinzugefügt,
– durch das Programm aktualisiert und
– überschrieben oder
– gelöscht werden.

Die folgende Tabelle stellt die Möglichkeiten für OPEN I-O zusammen:

Anweisung	Eintrag in der ACCESS MODE-Klausel		
	SEQUENTIAL	RANDOM	DYNAMIC
START	START... [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-START]	Anweisung nicht zulässig	START... [KEY IS...] [INVALID KEY...] [NOT INVALID KEY] [END-START]
READ	READ...[NEXT] [INTO...] [AT END...] [NOT AT END...] [END-READ...]	READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]	Für sequentiellen Zugriff: READ...NEXT [INTO...] [AT END...] [NOT AT END...] [END-READ]
			Für wahlfreien Zugriff: READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]
REWRITE	REWRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-REWRITE]	REWRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-REWRITE]	REWRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-REWRITE]
WRITE	Anweisung nicht zulässig	WRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-WRITE]	WRITE... [FROM...] [INVALID KEY...] [NOT INVALID KEY...] [END-WRITE]
DELETE	DELETE... [END-DELETE]	DELETE... [INVALID KEY...] [NOT INVALID KEY...] [END-DELETE]	DELETE... [INVALID KEY...] [NOT INVALID KEY...] [END-DELETE]

Tabelle 9-18: Erlaubte Ein-/Ausgabeanweisungen für OPEN I-O

OPEN I-O setzt voraus, daß die zu verarbeitende Datei bereits existiert. Es ist daher nicht möglich, in diesem Modus eine indizierte Datei neu zu erstellen.

Welche der oben erwähnten Verarbeitungsformen durchgeführt werden können und wie die Ein-/Ausgabeanweisungen dabei wirken, hängt von der vereinbarten Zugriffsart ab:

– ACCESS MODE IS SEQUENTIAL

  erlaubt es, wie bei OPEN INPUT die Datei mit READ sequentiell zu lesen und dabei durch einen vorhergehenden START auf einen beliebigen Satz der Datei als Anfangspunkt zu positionieren.

  Darüberhinaus kann nach einem erfolgreichen READ der gelesene Satz

  – durch das Programm aktualisiert und mit REWRITE zurückgeschrieben oder
  – mit DELETE logisch gelöscht werden.

  Dabei darf zwischen READ und REWRITE bzw. DELETE

  – keine weitere Ein-/Ausgabeanweisung für diese Datei ausgeführt und
  – das RECORD KEY-Schlüsselfeld nicht verändert werden.

– ACCESS MODE IS RANDOM

  ermöglicht es, wie bei OPEN INPUT mit READ Sätze wahlfrei zu lesen.

  Ferner können mit WRITE neue Sätze in die Datei eingefügt und mit REWRITE bzw. DELETE bereits in der Datei vorhandene Sätze überschrieben bzw. gelöscht werden (unabhängig davon, ob sie vorher gelesen wurden). Das RECORD KEY-Schlüsselfeld muß dazu vor jeder WRITE-, REWRITE- oder DELETE-Anweisung mit dem Schlüssel des Satzes versorgt werden, der hinzugefügt, überschrieben oder gelöscht werden soll.

  Wird bei WRITE der Schlüssel eines bereits vorhandenen Satzes bzw. bei REWRITE oder DELETE der Schlüssel eines nicht existierenden Satzes angegeben, dann tritt eine INVALID KEY-Bedingung auf und WRITE oder REWRITE bzw. DELETE verzweigt zur INVALID KEY-Anweisung oder zur vereinbarten USE-Pozedur.

– ACCESS MODE IS DYNAMIC

  gestattet es, die Datei sowohl sequentiell als auch wahlfrei zu verarbeiten. Die jeweilige Zugriffsart wird dabei über das Format der READ-Anweisung gewählt.

9.4.5 Positionieren mit START

In indizierten (wie auch in relativen) Dateien kann mit START auf jeden beliebigen Datensatz als Ausgangspunkt für nachfolgende sequentielle Leseoperationen positioniert werden. Den Schlüssel (Primär- oder Sekundärschlüssel) des zuerst zu lesenden Satzes legt START dabei über eine Vergleichsbedingung fest.

Das folgende Beispiel zeigt, wie es mit Hilfe der Spracherweiterung (gegenüber ANS85) START...KEY LESS... möglich ist, eine indizierte Datei sequentiell in umgekehrter Richtung zu verarbeiten; d.h. in der Reihenfolge absteigender Satzschlüssel, beginnend mit dem höchsten in der Datei vorhandenen Schlüssel:

Beispiel 9-11: Verarbeiten einer indizierten Datei in umgekehrter Richtung

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  INDREV.
*    INDREV VERARBEITET DIE SAETZE EINER INDIZIERTEN DATEI
*    IN DER FOLGE ABSTEIGENDER SATZSCHLUESSEL.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     TERMINAL IS T.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT IND-DATEI
     ASSIGN TO "INDFILE"
     ORGANIZATION IS INDEXED
     ACCESS IS DYNAMIC
     RECORD KEY IS REC-KEY.
 DATA DIVISION.
 FILE SECTION.
 FD  IND-DATEI.
 01  IND-SATZ.
     05  REC-KEY                PIC X(8).
     05  REC-TEXT               PIC X(72).
 WORKING-STORAGE SECTION.
 01  VERARBEITUNGS-SCHALTER     PIC X.
     88  VERARBEITUNGS-ENDE               VALUE "1".
 PROCEDURE DIVISION.
 VORLAUF.
     OPEN I-O IND-DATEI ------------------------------------------------ (1)
     MOVE HIGH-VALUE TO REC-KEY ---------------------------------------- (2)
     MOVE "0" TO VERARBEITUNGS-SCHALTER.
 DATEI-VERARBEITEN.
     START IND-DATEI KEY LESS OR EQUAL REC-KEY
        INVALID KEY
           DISPLAY "DATEI IST LEER" UPON T
           SET VERARBEITUNGS-ENDE TO TRUE
        NOT INVALID KEY
           READ IND-DATEI NEXT ------------------------------------------ (3)
              AT END
                 SET VERARBEITUNGS-ENDE TO TRUE
```

```
               NOT AT END
                  DISPLAY "HOECHSTER SATZSCHLUESSEL: " REC-KEY
                     UPON T
                  PERFORM SATZ-VERARBEITEN
            END-READ
      END-START

      PERFORM WITH TEST BEFORE UNTIL VERARBEITUNGS-ENDE
         START IND-DATEI KEY LESS REC-KEY ------------------------------- (4)
            INVALID KEY
               SET VERARBEITUNGS-ENDE TO TRUE
            NOT INVALID KEY
               READ IND-DATEI NEXT
                  AT END
                     SET VERARBEITUNGS-ENDE TO TRUE
                  NOT AT END
                     DISPLAY "NAECHSTER SATZSCHLUESSEL: " REC-KEY
                        UPON T
                     PERFORM SATZ-VERARBEITEN
               END-READ
         END-START
      END-PERFORM.
  NACHLAUF.
      CLOSE IND-DATEI
      STOP RUN.
  SATZ-VERARBEITEN.
 *
 *    VERARBEITUNG DES AKTUELLEN SATZES ------------------------------- (5)
 *
```

(1) Für die Verarbeitung wird die Datei IND-DATEI mit OPEN I-O eröffnet.

(2) Um den Satz mit dem höchsten Schlüssel in der Datei zu erhalten, wird
- der RECORD KEY mit dem höchstmöglichen Wert (HIGH-VALUE im NATIVE-Alphabet) vorbelegt und
- mit START...KEY LESS OR EQUAL positioniert.

(3) READ...NEXT liest den Satz ein, auf den zuvor mit START positioniert wurde.

(4) START...KEY LESS positioniert auf den Vorgänger des zuletzt gelesenen Satzes.

(5) Der eingelesene Satz wird verarbeitet. Falls sein RECORD KEY dabei verändert wird, muß dessen ursprünglicher Wert vor der folgenden START-Anweisung wiederhergestellt werden.

9.4.6 Ein-/Ausgabezustände

Jeder Datei im Programm können mit der FILE STATUS-Klausel Datenfelder zugeordnet werden, in denen das Laufzeitsystem nach jedem Zugriff auf die Datei Informationen darüber hinterlegt,

– ob die Ein-/Ausgabeoperation erfolgreich war und

– welcher Art ggf. die dabei aufgetretenen Fehler sind.

Diese Informationen können z.B. in den DECLARATIVES durch USE-Prozeduren ausgewertet werden und gestatten eine Analyse von Ein-/Ausgabefehlern durch das Programm. Als Erweiterung zum COBOL-Standard bietet COBOL85 die Möglichkeit, in diese Analyse durch die Schlüssel der DVS-Fehlermeldungen einzubeziehen. Dadurch läßt sich eine feinere Differenzierung der Fehlerursachen erreichen.

Die FILE STATUS-Klausel wird im FILE-CONTROL-Paragraphen der ENVIRONMENT DIVISION angegeben; ihr Format ist (siehe [1]):

```
FILE STATUS IS datenname-1 [datenname-2]
```

Dabei müssen datenname-1 und, falls angegeben, datenname-2 in der WORKING-STORAGE SECTION oder der LINKAGE SECTION definiert sein. Für die Formate und die möglichen Werte dieser beiden Datenfelder gelten folgende Regeln:

datenname-1

– muß als zwei Byte langes numerisches oder alphanumerisches Datenfeld erklärt werden, also z.B.

```
01 datenname-1          PIC X(2).
```

– enthält nach jeder Ein-/Ausgabeoperation auf die zugeordnete Datei einen zweistelligen numerischen Zustandscode, dessen Bedeutung der Tabelle am Ende dieses Abschnitts entnommen werden kann.

datenname-2

– muß als sechs Byte langes Gruppenfeld der folgenden Struktur erklärt werden:

```
01 datenname-2.
   02 datenname-2-1       PIC 9(2) COMP.
   02 datenname-2-2       PIC X(4).
```

– dient der Aufnahme des DVS-Fehlerschlüssels (DVS-Codes) zum jeweiligen Ein-/ Ausgabezustand und enthält nach jedem Zugriff auf die zugeordnete Datei einen Wert, der vom Inhalt des Feldes datenname-1 abhängt und sich aus folgender Zusammenstellung ergibt:

Inhalt von datenname-1 ungleich 0?	DVS-Code ungleich 0?	Wert von datenname-2-1	Wert von datenname-2-2
nein	nicht relevant	undefiniert	undefiniert
ja	nein	0	undefiniert
ja	ja	64	DVS-Code der zugeordneten Fehlermeldung

Die DVS-Codes und die zugeordneten Fehlermeldungen können Handbuch [4] entnommen werden.

Tabelle 9-19: Ein-/Ausgabezustände für indizierte Dateien

Ein-/Ausgabe-Zustand	Bedeutung
	Erfolgreiche Ausführung
00	Die Ein-/Ausgabe-Anweisung wurde erfolgreich ausgeführt. Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar.
02	Ein Satz wurde über ALTERNATE KEY gelesen, und es existiert bei sequentiellem Weiterlesen über denselben Schlüssel noch mindestens ein Nachfolgesatz mit identischem Schlüsselwert. Ein Satz mit ALTERNATE KEY WITH DUPLICATES wurde geschrieben, und es gibt bereits für mindestens einen Alternativschlüssel einen Satz mit identischem Schlüsselwert.
04	Satzlängenkonflikt: Eine READ-Anweisung wurde erfolgreich ausgeführt. Die Länge des gelesenen Datensatzes liegt jedoch nicht in den Grenzen, die durch die Satzbeschreibunge der Datei festgelegt wurden.
05	OPEN-Anweisung auf eine nicht vorhandene OPTIONAL-Datei
	Erfolglose Ausführung: Endebedingung
10	Es wurde versucht, ein sequentielles READ auszuführen. Es war jedoch kein nächster logischer Datensatz vorhanden, da das Dateiende erreicht war.
	Erfolglose Ausführung: Schlüsselfehlerbedingung
21	Reihenfolgefehler für eine Datei bei ACCESS MODE IS SEQUENTIAL: 1. Der Wert des Satzschlüssels wurde zwischen der erfolgreichen Ausführung einer READ-Anweisung und der Ausführung der nachfolgenden REWRITE-Anweisung geändert 2. Bei aufeinanderfolgenden WRITE-Anweisungen wurde die aufsteigende Folge von Satzschlüsseln nicht eingehalten.
22	Doppelter Schlüssel Es wurde versucht, eine WRITE-Anweisung mit einem Primärschlüssel auszuführen, für den innerhalb der Datei bereits ein Satz vorhanden ist. Es wurde versucht, einen Satz mit ALTERNATE KEY ohne WITH DUPLICATES-Angabe zu erstellen, obwohl in der Datei bereits ein Alternativschlüssel mit identischem Schlüsselwert vorhanden ist.
23	Datensatz nicht gefunden Es wurde versucht, anhand eines Schlüssels mit einer READ-, START-, DELETE- oder REWRITE-Anweisung auf einen Datensatz zuzugreifen, der in der Datei nicht vorhanden ist.
24	Überschreiten der Bereichsgrenzen Es wurde versucht, eine WRITE-Anweisung außerhalb der vom System festgelegten Bereichsgrenzen einer indizierten Datei auszuführen.

Ein-/Ausgabe-Zustand	Bedeutung
	Erfolglose Ausführung: Permanenter Fehler
30	Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar (der DVS-Code liefert weitere Informationen).
35	Es wurde versucht, eine OPEN INPUT, I-O- oder EXTEND-Anweisung für eine nicht optionale Datei auszuführen, die nicht vorhanden war.
37	OPEN-Anweisung auf eine Datei, die wegen folgender Bedingungen nicht eröffnet werden kann: 1. OPEN OUTPUT/I-O/EXTEND auf eine schreibgeschützte Datei (Paßwort, RETENTION-PERIOD, ACCESS=READ) 2. OPEN INPUT auf eine lesegeschützte Datei (Paßwort)
38	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die vorher mit der LOCK-Angabe geschlossen wurde.
39	Die OPEN-Anweisung war aus einem der folgenden Gründe erfolglos: 1. Im SET-FILE-LINK-Kommando wurden einer oder mehrere der Operanden ACCESS-METHOD, RECORD-FORMAT, RECORD-SIZE oder KEY-LENGTH mit Werten angegeben, die von den entsprechenden expliziten oder impliziten Programmangaben abweichen. 2. Bei einer Eingabedatei trat ein Satzlängenfehler auf (Katalogüberprüfung, falls RECFORM=F). 3. Die Satzlänge ist größer als die BLKSIZE-Angabe im Katalog einer Eingabedatei. 4. Für eine Eingabedatei stimmt der Katalogeintrag eines der Operanden FCBTYPE, RECFORM, RECSIZE (falls RECFORM=F), KEYPOS oder KEYLEN nicht mit den entsprechenden expliziten oder impliziten Programmangaben bzw. mit den entsprechenden Angaben im SET-FILE-LINK-Kommando überein. 5. Es wurde versucht, eine Datei zu eröffnen, deren Alternativschlüssel nicht mit den im Programm angegebenen Schlüsselwerten der ALTERNATE RECORD KEY-Klausel übereinstimmen.
	Erfolglose Ausführung: Logischer Fehler
41	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die bereits eröffnet ist.
42	Es wurde versucht, eine CLOSE-Anweisung für eine Datei auszuführen, die nicht eröffnet ist.
43	Bei ACCESS MODE IS SEQUENTIAL: Die letzte vor Ausführung einer DELETE- oder REWRITE-Anweisung ausgeführte Ein-/Ausgabe-Anweisung war keine erfolgreiche READ-Anweisung.
44	Überschreiten der Satzlängengrenzen: Es wurde versucht, eine WRITE- oder REWRITE-Anweisung auszuführen. Die Länge des Datensatzes liegt jedoch nicht in dem für diese Datei zulässigen Bereich.

Ein-/Ausgabe-Zustand	Bedeutung
46	Es wurde versucht, ein sequentielles READ für eine Datei auszuführen, die sich im Eröffnungsmodus INPUT oder I-O befindet; ein nächster gültiger Datensatz steht aber nicht zur Verfügung. Grund: 1. Die vorhergehende START-Anweisung war erfolglos, oder 2. Die vorhergehende READ-Anweisung war erfolglos, ohne die Endebedingung zu verursachen, oder 3. Es wurde versucht, nach bereits erkannter AT END-Bedingung eine READ-Anweisung auszuführen.
47	Es wurde versucht, eine READ- oder START-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus INPUT oder I-O befindet.
48	Es wurde versucht, eine WRITE-Anweisung für eine Datei auszuführen, die sich – bei sequentiellem Zugriff nicht im Eröffnungsmodus OUTPUT oder EXTEND, – bei wahlfreiem oder dynamischem Zugriff nicht im Eröffnungsmodus OUTPUT oder I-O befindet.
49	Es wurde versucht, eine DELETE- oder REWRITE-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus I-O befindet.
	Sonstige erfolglose Ausführungen
90	Systemfehler; es ist keine weitere Information über die Ursache vorhanden.
91	OPEN-Fehler; die eigentliche Ursache ist aus dem DVS-Code ersichtlich (siehe "FILE-STATUS-Klausel" mit Angabe von datenname-2).
93	Nur bei Simultanverarbeitung (siehe Abschnitt 9.5): Die Ein-/Ausgabe-Anweisung konnte nicht erfolgreich durchgeführt werden, weil ein anderer Prozeß auf dieselbe Datei zugreift und die Zugriffe nicht vereinbar sind.
94	1. Nur bei Simultanverarbeitung (siehe Abschnitt 9.5): Die Aufruffolge READ - REWRITE/DELETE wurde nicht eingehalten. 2. Die Satzlänge ist größer als die Blocklänge.
95	Unverträglichkeit zwischen den Angaben im BLOCK-CONTROL-INFO- oder BUFFER-LENGTH-Operanden des SET-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder dem Format des verwendeten Datenträgers

9.5 Simultanverarbeitung von Dateien (SHARED-UPDATE)

9.5.1 ISAM-Dateien

ISAM-Dateien mit indizierter oder relativer Dateiorganisation können für mehrere Benutzer gleichzeitig zugänglich gemacht werden. Dies geschieht mit dem Operanden SHARED-UPDATE im SUPPORT-Parameter des SET-FILE-LINK-Kommandos:

```
/SET-FILE-LINK linkname,dateiname,SUPPORT=DISK(SHARED-UPDATE=YES)
```

Die folgende Tabelle zeigt, welche OPEN-Anweisungen für einen Benutzer B möglich sind, nachdem die Datei von Benutzer A bereits eröffnet wurde.

		Zulässige Angaben für Benutzer B					
Von Benutzer A gewählte Angaben	**OPEN-Anweisung**	SHARED-UPDATE=YES			SHARED-UPDATE=NO		
		INPUT	I-O	OUPUT/ EXTEND	INPUT	I-O	OUTPUT/ EXTEND
SHARED-UPDATE=YES	INPUT	X	X		X		
	I-O	X	X				
	OUTPUT / EXTEND						
SHARED-UPDATE=NO	INPUT	X			X		
	I-0						
	OUTPUT / EXTEND						

Tabelle 9-20: Zulässige OPEN-Anweisungen bei Simultanverarbeitung

X zulässige Kombinationen von OPEN-Anweisung und SHARED-UPDATE-Angabe

Aus der Tabelle geht hervor, daß die Angabe von SHARED-UPDATE=YES für INPUT-Dateien überflüssig ist, falls alle Anwender OPEN INPUT verwenden.
Wenn SHARED-UPDATE=YES für INPUT-Dateien trotzdem angegeben werden muß, da mindestens ein Anwender OPEN I-O verwendet, wird das nachfolgend beschriebene Sperren bzw. Entsperren nicht durchgeführt.

Die Angabe SHARED-UPDATE=YES ist nur für die gleichzeitige Aktualisierung von einer oder mehreren ISAM-Dateien (OPEN I-O) durch zwei oder mehr Dialogbenutzer sinnvoll und auch notwendig.

Aktualisierungen im Stapelbetrieb sollten nacheinander ablaufen, um sowohl Logikfehler als auch Laufzeitverlängerungen zu vermeiden (unnötige Angabe von SHARED-UPDATE=YES kostet Laufzeit und CPU-Zeit).

Bei Angabe von SHARED-UPDATE=YES wird automatisch auch WRITE-CHECK=YES gesetzt, d.h. die ISAM-Puffer werden nach jeder Änderung sofort zurückgeschrieben. Dies ist aus Datensicherheits- und Eindeutigkeitsgründen erforderlich, erhöht aber wesentlich die Anzahl der Ein-/Ausgaben.

Um Datenkonsistenz bei gleichzeitiger Aktualisierung einer ISAM-Datei durch mehrere Benutzer zu gewährleisten, benutzt das COBOL85-Laufzeitsystem den Sperr- und Entsperrmechanismus der DVS-Zugriffsmethode ISAM. Dieser Mechanismus sorgt für das Sperren bzw. Entsperren der Datenblöcke, in denen die durch die Anweisungen READ, WRITE, REWRITE oder DELETE angesprochenen Datensätze liegen.

Ein Datenblock ist das Vielfache einer PAM-Seite (2048 byte), das durch den BUFFER-LENGTH-Parameter im SET-FILE-LINK-Kommando implizit oder explizit beim Erzeugen der Datei vereinbart wurde (siehe 9.1.1).

Ist im folgenden von Datensatzsperre die Rede, ist immer die Sperre des ganzen Blocks, in dem dieser Datensatz liegt, gemeint.

Für die Simultanverarbeitung von ISAM-Dateien gibt es eine Formaterweiterung der READ- bzw. START-Anweisung, die jedoch nur wirksam wird, wenn im SET-FILE-LINK-Kommando SHARED-UPDATE=YES angegeben und die Datei mit OPEN I-O eröffnet ist.

Formaterweiterung für alle Formate:

```
READ  dateiname [WITH NO LOCK]...
START dateiname [WITH NO LOCK]...
```

Regeln für die Simultanaktualisierung

1. READ- oder START-Anweisung mit WITH NO LOCK-Zusatz:

   Gibt Benutzer A WITH NO LOCK an und ist der entsprechende Datensatz vorhanden, wird dieser gelesen bzw. wird auf diesen positioniert, ungeachtet einer etwa durch einen anderen Benutzer bereits gesetzten Sperre. Der Datensatz wird nicht gesperrt. Eine REWRITE- bzw. DELETE-Anweisung kann auf einen so gelesenen Satz nicht ausgeführt werden.

   Ein simultaner Benutzer B kann denselben Datensatz sowohl lesen als auch aktualisieren.

2. READ- oder START-Anweisung ohne WITH NO LOCK-Zusatz:

   Gibt Benutzer A WITH NO LOCK nicht an und ist der entsprechende Datensatz vorhanden, kann eine READ- bzw. START-Anweisung nur dann erfolgreich ausgeführt werden, wenn der entsprechende Datensatz nicht bereits durch Benutzer B gesperrt ist. War die Ausführung der Anweisung erfolgreich, wird der Datensatz gesperrt. Vor Aufhebung der Sperre kann Benutzer B denselben oder irgendeinen anderen Datensatz desselben Datenblocks nur mit WITH NO LOCK lesen oder auf ihn positionieren, er kann aber keinen Datensatz dieses Datenblocks aktualisieren. (Hat Benutzer B die Datei mit OPEN INPUT eröffnet, kann er Sätze des gesperrten Datenblocks immer lesen.)

3. Aktualisierung von Datensätzen:

   Soll durch eine REWRITE- oder DELETE-Anweisung ein Datensatz aktualisiert werden, muß der betroffene Datensatz unmittelbar zuvor durch eine READ-Anweisung (ohne WITH NO LOCK-Zusatz!) gelesen werden. Nach dieser READ- und vor der REWRITE- bzw. DELETE-Anweisung darf für dieselbe ISAM-Datei keine weitere Ein-/Ausgabeanweisung ausgeführt werden. Zwischen diesen beiden Anweisungen dürfen für andere ISAM-Dateien, deren SET-FILE-LINK-Kommando SHARED-UPDATE=YES enthält und die zur gleichen Zeit mit OPEN I-O eröffnet sind, nur READ- oder START-Anweisungen mit WITH NO LOCK-Zusatz ausgeführt werden. Anweisungen für andere ISAM-Dateien (ohne SHARED-UPDATE=YES und OPEN I-O) dürfen ausgeführt werden.
   Ein Verstoß gegen diese Vorschriften führt zu einer erfolglosen REWRITE- bzw. DELETE-Anweisung mit FILE STATUS 94.

4. Wartezeiten bei einer Sperre:

   Hat Benutzer A aufgrund einer erfolgreich ausgeführten READ- oder START-Anweisung einen Datensatz gesperrt und versucht Benutzer B auf denselben Datensatz oder irgendeinen anderen aus demselben Datenblock eine READ- oder START-Anweisung ohne WITH NO LOCK-Zusatz auszuführen, so führt dies für letzteren nicht sofort zum Mißerfolg. Benutzer B wird in eine Warteschlange eingeordnet, in der er auf die Freigabe der Sperre durch Benutzer A wartet. Erst wenn eine maximale Wartezeit abgelaufen und die Entsperrung innerhalb dieser Frist nicht erfolgt ist, gilt die Anweisung als erfolglos und FILE STATUS 93 wird gesetzt. Wurde die Sperre vor Ablauf der Wartezeit aufgehoben, so kann Benutzer B mit dem erfolgreichen Aufruf fortfahren.

5. Freigabe eines gesperrten Datensatzes:

   Ein Benutzer behält eine Datensatzsperre solange bei, bis er eine der folgenden Anweisungen ausführt:

   – erfolgreiche REWRITE- oder DELETE-Anweisung auf den gesperrten Datensatz

– WRITE-Anweisung auf eine ISAM-Datei, deren SET-FILE-LINK-Kommando SHARED-UPDATE=YES enthält und die mit OPEN I-O eröffnet ist (d.h. auf dieselbe Datei, die den gesperrten Datensatz enthält, oder auf eine andere ISAM-Datei; Entsperrung erfolgt auch bei Auftreten der INVALID KEY-Bedingung)

– READ- oder START-Anweisung mit WITH NO LOCK-Zusatz auf dieselbe Datei (Entsperrung erfolgt auch bei Auftreten der AT END- oder INVALID KEY-Bedingung)

– READ- oder START-Anweisung ohne WITH NO LOCK-Zusatz auf einen Datensatz innerhalb eines anderen Datenblocks derselben Datei (Entsperrung erfolgt auch bei Auftreten der AT END- oder INVALID KEY-Bedingung)

– READ- oder START-Anweisung ohne WITH NO LOCK-Zusatz auf eine andere ISAM-Datei, deren SET-FILE-LINK-Kommando SHARED-UPDATE=YES enthält und die mit OPEN I-O eröffnet ist (Entsperrung erfolgt auch bei Auftreten der AT END- oder INVALID KEY-Bedingung)

– CLOSE-Anweisung für dieselbe Datei.

Eine Anweisung für eine ISAM-Datei kann also die Aufhebung einer Datensatzsperre auf einer anderen ISAM-Datei bewirken.

Hinweise

1. Soll in einem Programm eine ISAM-Datei (SHARED-UPDATE=YES und OPEN I-O) bearbeitet werden, sollte für diese Datei eine USE AFTER STANDARD ERROR-Prozedur vorgesehen werden. In dieser Prozedur können die simultanverarbeitungsspezifischen FILE STATUS-Werte 93 (Datensatz von einem simultanen Benutzer gerade gesperrt) und 94 (REWRITE- oder DELETE-Anweisung ohne vorherige READ-Anweisung) abgefragt und angemessen verarbeitet werden.
2. Es sollte immer berücksichtigt werden, daß aufgrund des Block-Sperrmechanismus von ISAM beim Sperren eines Datensatzes auch zugleich alle Datensätze desselben Blocks für alle simultanen Benutzer gesperrt werden.
3. Für einen Benutzer kann höchstens ein Datenblock gesperrt, d.h., vor Aktualisierung durch andere Benutzer geschützt sein. Dies gilt auch dann, wenn er mehrere ISAM-Dateien (alle SHARED-UPDATE=YES) im Modus OPEN I-O eröffnet hat.
4. Eine "Deadlock"-Situation (gegenseitiges Sperren von Datenblöcken durch verschiedene Benutzer) ist ausgeschlossen, da für jeden Benutzer nur ein einziger Block aller ISAM-Dateien (alle SHARED-UPDATE=YES) gesperrt sein kann.Dies gilt jedoch nicht, wenn gleichzeitig auf eine PAM-Datei mit SHARED-UPDATE=YES im I-O-Modus zugegriffen wird!

5. Falls zwischen einer READ- und einer REWRITE- bzw. DELETE-Anweisung für einen Datensatz ein Zugriff auf einen anderen Datenblock derselben oder einer anderen ISAM-Datei erfolgt, der gleichzeitig eine Entsperrung des zuvor gesperrten Datenblocks zur Folge hat, muß der Datensatz vor Ausführung der REWRITE- bzw. DELETE-Anweisung noch einmal gelesen werden. Da der betroffene Datenblock in der Zwischenzeit für andere Benutzer entsperrt war, könnte der Inhalt des Datensatzes verändert worden sein (siehe Beispiel 9-11a).

   Erfolgt der Zugriff auf den anderen Datenblock bzw. die andere Datei ohne Sperrmechanismus, könnten die dadurch bereitgestellten Daten während der Verarbeitung bereits wieder durch simultane Benutzer verändert worden sein, ehe die REWRITE- bzw. DELETE-Anweisung ausgeführt worden ist (siehe Beispiel 9-11b).

6. Um zu vermeiden, daß ein Benutzer möglicherweise mit nicht mehr aktuellen Daten arbeitet, sollte der WITH NO LOCK-Zusatz nur dann verwendet werden, wenn dies unbedingt erforderlich ist.

7. Ein gesperrter Datensatz (Datenblock) führt bei simultanen Benutzern, die auf denselben Datensatz oder einen anderen desselben Blocks zugreifen wollen, zu Wartezeiten. Um diese so kurz wie möglich zu halten, sollte die Sperre sobald wie möglich wieder aufgehoben werden. Wird die Sperre nicht rechtzeitig aufgehoben, läuft die Wartezeit ab, und das Programm verzweigt in die vorgesehene USE-Prozedur, sofern vorhanden (siehe Beispiel 9-12) oder wird abgebrochen (mit Meldung COB9151, FILE STATUS 93 und DVS-Fehlerschlüssel DAAA).

Beispiel 9-12: Lesen und Zurückschreiben in Datei ISAM1, wenn vor dem Zurückschreiben Daten aus einer Datei ISAM2 benötigt werden

(a) Ohne WITH NO LOCK-Zusatz: zweimalige READ-Anweisung auf dieselbe Datei erforderlich, dafür Sperrzeiten kürzer:

```
...
READ ISAM1 INTO WORK1 ------------------------------------------------- (1)
  INVALID KEY...
...
READ ISAM2 ------------------------------------------------------------ (2)
  INVALID KEY...
...

Verarbeitung von WORK1 unter Berücksichtigung von ISAM2SATZ
...
READ ISAM1 ------------------------------------------------------------ (3)
  INVALID KEY...

Prüfung, ob ISAM1SATZ inzwischen geändert, gegebenenfalls erneute Verarbeitung
...
REWRITE ISAM1SATZ FROM WORK1 ------------------------------------------ (4)
  INVALID KEY...
```

(1) Lesen eines Datensatzes aus ISAM1 und Zwischenspeichern in WORK1, betroffener Datenblock in ISAM1 gesperrt

(2) Lesen eines Datensatzes aus ISAM2, Aufhebung der Sperre in ISAM1, Sperrung des betroffenen Datenblocks in ISAM2

(3) Erneutes Lesen des Datenatzes in ISAM1, Aufhebung der Sperre in ISAM2, Sperrung des betroffenen Datenblocks in ISAM1

(4) Zurückschreiben des Datensatzes nach ISAM1, Aufhebung der Sperre in ISAM1

(b) Mit WITH NO LOCK-Zusatz: nur eine READ-Anweisung auf dieselbe Datei erforderlich, dafür Sperrzeiten länger:

```
...
READ ISAM1 ------------------------------------------------------------ (1)
  INVALID KEY...
...
READ ISAM2 WITH NO LOCK ----------------------------------------------- (2)
  INVALID KEY...
...

Verarbeitung von ISAM1SATZ unter Berücksichtigung von ISAM2SATZ
...
REWRITE ISAM1SATZ FROM WORK1------------------------------------------- (3)
  INVALID KEY...
```

(1) Lesen eines Datensatzes aus ISAM1, betroffener Datenblock gesperrt

(2) Lesen eines Datensatzes aus ISAM2, betroffener Datenblock nicht gesperrt

(3) Zurückschreiben des Datensatzes nach ISAM1, Sperre wird aufgehoben

Beispiel 9-13: Verzweigen zu einer USE AFTER STANDARD ERROR-Prozedur

```
   ...
FILE-CONTROL.
    SELECT ISAM1

   ...
    FILE STATUS IS FILESTAT1.
WORKING-STORAGE SECTION.
77  FILESTAT1   PIC 99.
  ...
PROCEDURE DIVISION.
DECLARATIVES.
ISAM1ERR SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON ISAM1.
SPERRE.
    IF FILESTAT1 = 93
       THEN DISPLAY "SATZ ZUR ZEIT GESPERRT" UPON T
       ELSE DISPLAY "DMS-FEHLER ISAM1, FILE-STATUS="
                    FILESTAT1 UPON T.
ISAM1ERR-EX.
    EXIT. ------------------------------------------------------------------ (1)

END DECLARATIVES.
STEUER SECTION.
    ...
```

(1) Verzweigen auf die Anweisung hinter der fehlerverursachenden Anweisung. Wie der aufgetretene Fehler sinnvoll zu behandeln ist, muß für den jeweiligen Anwendungsfall entschieden werden.

9.5.2 PAM-Dateien

Eine Datei mit relativer Organisation und FCBTYPE=PAM kann - ebenso wie eine ISAM-Datei - von mehreren Benutzern simultan aktualisiert werden, wenn das SET-FILE-LINK-Kommando SHARED-UPDATE=YES enthält und die Datei mit OPEN I-O eröffnet ist.

Um Datenkonsistenz bei simultaner Aktualisierung zu ermöglichen, benutzt das COBOL85-Laufzeitsystem den Sperr- und Entsperrmechanismus der DVS-Zugriffsmethode UPAM. Die Zugriffskoordinierung erfolgt hier (anders als bei ISAM) dateispezifisch. Dies hat u.a. zur Folge, daß Anweisungen für eine Datei keine Auswirkungen auf eine andere Datei haben.

Die Sperrung betrifft - wie bei ISAM - nicht einen einzelnen Datensatz, sondern den gesamten Datenblock, in dem sich der Datensatz befindet (siehe "Indizierte Dateien").

Wie für ISAM-Dateien gibt es auch für PAM-Dateien (nur mit SHARED-UPDATE=YES, OPEN I-O) für alle Formate der READ- bzw. START-Anweisung die Erweiterung WITH NO LOCK (Beschreibung siehe "Indizierte Dateien").

Regeln für die Simultanaktualisierung

1. Das Lesen und Positionieren ohne bzw. mit WITH NO LOCK-Zusatz erfolgt wie bei ISAM-Dateien.
2. Aktualisierung von Datensätzen

   Soll durch eine REWRITE- bzw. DELETE-Anweisung ein Datensatz aktualisiert werden, muß der betroffene Datensatz (wie bei ISAM-Dateien) unmittelbar zuvor durch eine READ-Anweisung (ohne WITH NO LOCK-Zusatz) gelesen werden. Zwischen beiden Anweisungen darf für dieselbe Datei keine weitere Anweisung ausgeführt werden. Anweisungen für andere PAM-Dateien sind jedoch - anders als bei ISAM-Dateien zulässig (aufgrund der dateispezifischen Zugriffskoordinierung).
3. Wartezeiten bei einer Sperre

   Die maximale Wartezeit auf die Freigabe eines gesperrten Blocks beträgt 999 Sekunden. Nach Ablauf dieser Zeit wird, falls vorhanden, die USE AFTER STANDARD ERROR-Prozedur angesprungen oder das Programm mit der Fehlermeldung COB9151 beendet (FILE STATUS 93 und DVS-Fehlerschlüssel D9B0 oder D9B1).
4. Freigabe eines gesperrten Datensatzes

   Die Entsperrung eines gesperrten Datenblocks kann mit denselben Anweisungen bewirkt werden wie bei ISAM-Dateien, jedoch müssen sich alle Anweisungen auf dieselbe Datei beziehen.

   Im Unterschied zu ISAM-Dateien bewirkt also eine Anweisung für eine PAM-Datei keine Entsperrung von Datenblöcken einer anderen PAM-Datei.

Hinweise

1. Soll in einem Programm eine PAM-Datei (mit SHARED-UPDATE=YES, OPEN I-O) verarbeitet werden, sollte für diese Datei eine USE AFTER STANDARD ERROR-Prozedur vereinbart werden (siehe "Indizierte Dateien").
2. Anders als bei ISAM-Dateien (mit SHARED-UPDATE=YES, OPEN I-O) kann bei simultaner Verarbeitung mehrerer Dateien (alle mit SHARED-UPDATE=YES, OPEN I-O), von denen mindestens eine eine PAM-Datei ist, für jeden Benutzer je ein Datensatz in beliebig vielen Dateien gleichzeitig gesperrt (!) werden (innerhalb einer Datei immer nur ein Satz). Dadurch kann es zu sogenannten "Deadlock"-Situationen kommen (siehe Beispiel 9-13).
3. Wie bei ISAM-Dateien sollte auch bei PAM-Dateien die Sperre auf Datensätzen (Datenblöcken!) so schnell wie möglich aufgehoben werden, um die damit verbundenen Wartezeiten für andere Benutzer möglichst kurz zu halten.

Beispiel 9-14: "Deadlock"

Benutzer A:	Benutzer B:
`READ datei1 (satz n)`	`READ datei2 (satz m)`
`.`	`.`
`.`	`.`
`READ datei2 (satz m)`	`READ datei1 (satz n)`
(Block in datei1 nicht entsperrt)	(Block in datei2 nicht entsperrt)

Beide Benutzer warten auf Freigabe des jeweiligen Blocks ("Deadlock").

Die maximale Wartezeit auf die Freigabe eines gesperrten Blocks beträgt 999 Sekunden. Nach Ablauf dieser Zeit wird, falls vorhanden, die USE AFTER STANDARD ERROR-Prozedur angesprungen oder das Programm mit der Fehlermeldung COB9151 beendet (FILE STATUS 93 und DVS-Fehlerschlüssel D9B0 oder D9B1).

10 Sortieren und Mischen

10.1 COBOL-Sprachmittel zum Sortieren und Mischen

Das Sortieren und Mischen unterstützt COBOL85 durch folgende Sprachmittel (siehe [1]):

– Die Angabe des Literals "SORTWK" in der ASSIGN-Klausel

  Sie vereinbart explizit den Linknamen SORTWK für die Sortierdatei.

  Das Format der ASSIGN-Klausel für Sortierdateien läßt auch andere Angaben zu, die jedoch vom Compiler als Kommentar betrachtet werden. Der Linkname für die Sortierdatei ist stets SORTWK.

– Die Sortierdateierklärung (SD) in der DATA DIVISION

  Sie entspricht der Dateierklärung (FD) für andere Dateien und legt die physische Struktur, das Format und die Größe der Datensätze fest.

– Die Anweisungen SORT und MERGE in der PROCEDURE DIVISION

  SORT sortiert Datensätze nach einem oder mehreren (bis zu 64) Datenfeldern, die als Sortierschlüssel vereinbart wurden.
  Diese Datensätze können SORT aus einer Datei oder über eine Eingabeprozedur zur Verfügung gestellt werden. Die sortierten Sätze werden in eine Datei geschrieben oder einer Ausgabeprozedur übergeben.
  Für das Sortieren verwendet COBOL85 die Sortierfunktion des BS2000 SORT (siehe [6]).

  MERGE mischt in einer Sortierdatei Datensätze aus zwei oder mehreren gleichartig sortierten Eingabedateien anhand einer Anzahl von (bis zu 64) Datenfeldern, die als Sortierschlüssel vereinbart wurden.
  Die gemischten Sätze werden in eine Datei geschrieben oder einer Ausgabeprozedur übergeben.

– Die Vereinbarung von Eingabe- und Ausgabeprozeduren

  Eine Eingabeprozedur (INPUT PROCEDURE) kann für jede SORT-Anweisung vereinbart werden. Sie erlaubt es, die zu sortierenden Datensätze zu erzeugen oder zu bearbeiten, bevor sie über eine RELEASE-Anweisung an die Sortierdatei übergeben werden.

  Eine Ausgabeprozedur (OUTPUT PROCEDURE) kann für jede SORT- oder MERGE-Anweisung vereinbart werden. Sie erlaubt es, die sortierten bzw. gemischten Datensätze weiter zu bearbeiten, nachdem sie ihr über eine RETURN-Anweisung zur Verfügung gestellt wurden.

Hinweis

Durch Übersetzung mit der SDF-Option
RUNTIME-OPTIONS=PAR(SORTING-ORDER=BY-DIN) bzw.
mit COMOPT SORT-EBCDIC-DIN=YES
kann für alle SORT-Anweisungen eines Programms das Sortierformat ED des Dienstprogrammes SORT gewählt werden (siehe Handbuch [6]). Dies ermöglicht eine Textsortierung nach DIN-Norm für EBCDIC. Dabei werden

– Kleinbuchstaben den entsprechenden Großbuchstaben gleichgesetzt,

– die Zeichen

  "ä" / "Ä" mit "AE",
  "ö" / "Ö" mit "OE",
  "ü" / "Ü" mit "UE" sowie
  "ß" mit "SS" identifiziert sowie

– die Ziffern vor den Buchstaben einsortiert.

10.2 Dateien für das Sortierprogramm

Für einen Sortiervorgang werden folgende Dateien benötigt:

Sortierdatei

In dieser Datei (Arbeitsbereich) werden Datensätze sortiert. Ihr Name wird z.B. vereinbart über die Klausel

```
SELECT sortierdatei ASSIGN TO "SORTWK"
```

Außerdem muß diese Datei in der Sortierdateierklärung (SD) der DATA DIVISION beschrieben sein. Mit der Anweisung

```
SORT sortierdatei ...
```

wird auf diese Datei zugegriffen.

Ohne daß der Benutzer ein SET-FILE-LINK-Kommando angibt, wird diese Datei unter dem Namen `SORTWORK.tsn.jjmmtt.hhmmss` (tsn Prozessfolgenummer, jj Jahresangabe, mm Monatsangabe, tt Tagesangabe, hhmmss sechsstellige Uhrzeitangabe) katalogisiert. Der Linkname ist SORTWK. Nach normalem Sortierende wird diese Datei gelöscht.

Die Größe der Sortierdatei beim Einrichten ohne SET-FILE-LINK-Kommando beträgt standardmäßig 24 x 16 = 384 PAM-Seiten (durch Versorgen von SORT-Sonderregistern kann dieser Wert beeinflußt werden). Demnach ist die Primärzuweisung 384 PAM-Seiten. Die Sekundärzuweisung ist 1/4 davon, also 96 PAM-Seiten.

Mit dem Kommando

```
MODIFY-FILE-ATTRIBUTES dateiname,
           SUPPORT=PUBLIC-DISK(SPACE=RELATIVE(PRIMARY-ALLOCATION=..,
                      SECONDARY-ALLOCATION=...))
```

kann der Benutzer die Größe der Sortierdatei selbst bestimmen (siehe Handbuch [6]). Empfehlenswert ist dies bei großen Dateien. Nach normalem Sortierende wird diese Datei geschlossen, aber **nicht** gelöscht.

SORT-Sonderregister (siehe [1]):

Vor dem Sortieren kann der Programmierer folgende SORT-Sonderregister versorgen:

– SORT-FILE-SIZE: mit der Anzahl der Sätze.

– SORT-MODE-SIZE: mit der durchschnittlichen Satzlänge.

Diese beiden Register verwendet das Dienstprogramm SORT zur Berechnung der Dateigröße, d.h., der Programmierer kann indirekt den SPACE-Operanden beeinflussen.

– SORT-CORE-SIZE: mit der gewünschten Größe der internen Arbeitsbereiche in byte.

  Durch diese Angaben kann der Programmablauf beeinflußt werden.

  Bei fehlender Angabe werden standardmäßig 24 x 4096 byte, d.h. 24 Seiten zu je 4 Kbyte angenommen. Näheres siehe [6], Optimierung von Sortierläufen.

Nach SORT- und RELEASE- und vor RETURN-Anweisungen kann der Programmierer das SORT-Sonderregister SORT-RETURN abfragen:

"0" zeigt das ordnungsgemäße Sortieren an,

"1" das fehlerhafte Sortieren.

Diese Abfrage empfiehlt sich, da bei fehlerhaftem Sortieren der Programmlauf nicht abgebrochen wird.

Die fehlerhafte Belegung eines SORT-Sonderregisters bewirkt die Fehlermeldung COB9134 (siehe Kap.14).

Eingabedatei(en)

Ist keine Eingabeprozedur definiert, generiert COBOL85 einen OPEN INPUT und einen READ...AT END für die angegebene Datei. Jede Eingabedatei muß im COBOL-Programm definiert sein.

Die Linknamen SORTIN und SORTINnn ($01 \leq nn \leq 99$) dürfen nicht innerhalb eines Sortierprogramms verwendet werden.

Ausgabedatei

Ist keine Ausgabeprozedur definiert, generiert COBOL85 einen OPEN OUTPUT und einen WRITE für die angegebene Datei. Die Ausgabedatei muß im COBOL-Programm definiert sein.

Der Linkname SORTOUT darf nicht innerhalb eines Sortierprogramms verwendet werden.

10.3 Fixpunktausgabe für Sortierprogramme und Wiederanlauf

Die Angabe der RERUN-Klausel (Format 2) veranlaßt die Ausgabe spezieller Fixpunkte für Sortierdateien. Fixpunkte enthalten Informationen über den Zustand des Sortiervorgangs. Sie sind notwendig, um ein vom Benutzer oder wegen Anlagenstörung abgebrochenes Programm wieder starten zu können, ohne den gesamten bisherigen Programmablauf wiederholen zu müssen. Die Ausgabe von Sortier-Fixpunkten empfiehlt sich vor allem bei großen Mengen von zu sortierenden Daten, da auf diese Weise eine erfolgte Vorsortierung bei einem Programmabbruch nicht verlorengeht.

Format 2 der RERUN-Klausel:

```
RERUN ON herstellername EVERY SORT OF sortierdateiname
```

herstellername: SYSnnn (000 ≤ nnn ≤ 200)

Fixpunkte werden in eine Fixpunktdatei (siehe Kap. 11) ausgegeben, die vom Sortierprogramm mit dem Standard-Dateinamen `SORTCKPT.Djjttt.Tnnnn` (jj= Jahr, ttt=laufender Tag des Jahres, nnnn=TSN des laufenden Prozesses) und mit dem Standard-Linknamen SORTCKPT eingerichtet wird (siehe [6]). Über den SPACE-Operanden im SUPPORT-Parameter des MODIFY-FILE-ATTRIBUTES-Kommandos kann der Anwender die Größe dieser Datei selbst bestimmen. Die Fixpunktausgabe wird auf SYSOUT protokolliert (Meldung EXC0301; siehe Kap.11). Den Zeitpunkt der Fixpunktausgabe kann der Anwender nicht selbst bestimmen.

Bei normaler Beendigung des Sortiervorgangs wird die Fixpunktdatei geschlossen, freigegeben und gelöscht, so daß der Benutzer keinen Zugriff auf sie hat.

Wird ein Sortierprogramm fehlerhaft abgebrochen, so kann man den Lauf beim zuletzt geschriebenen Fixpunkt wieder starten: Mit Hilfe der auf SYSOUT protokollierten Informationen gibt man dazu das RESTART-PROGRAM-Kommando (siehe Kap.11 und [3]).

10.4 Sortieren von Tabellen

Die BS2000-Sortierfunktion SORT läßt sich auch für das Sortieren von Tabellen verwenden. Als COBOL-Sprachmittel steht die SORT-Anweisung zur Verfügung (siehe COBOL85-Sprachbeschreibung [1]).

11 Fixpunktausgabe und Wiederanlauf

Fixpunkte werden von COBOL85-Objekten in eine externe Fixpunktdatei ausgegeben (ggf. zwei Fixpunktdateien, siehe unten). Ein Fixpunkt umfaßt Kennungsinformationen, Programmzustand, dazu bezogenen Systemzustand und virtuelle Speicherinhalte. Dies wird für einen möglichen späteren Wiederanlauf benötigt.

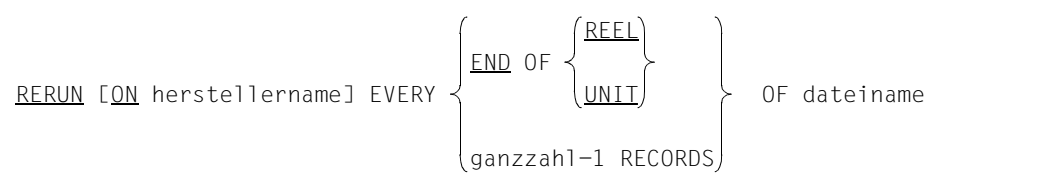
Durch Ausgabe solcher Fixpunkte kann ein absichtlich oder wegen Anlagenstörung abgebrochenes Programm zu beliebiger Zeit an der Stelle fortgesetzt werden, an der ein Fixpunkt ausgegeben wurde. Die Ausgabe von Fixpunkten empfiehlt sich vor allem bei Programmen mit langer Laufzeit; sie ist jedoch nur dann sinnvoll, wenn die Wiederherstellung der Ausgangsdaten bei einem eventuellen Wiederanlauf möglich ist.

Diese Funktionalität steht im POSIX-Subsystem nicht zur Verfügung (siehe Kapitel 13).

11.1 Fixpunktausgabe

Die Ausgabe von Fixpunkten veranlaßt der Benutzer mit der RERUN-Klausel. Dabei kann er den Zeitpunkt der Fixpunktausgabe bestimmen; eine Ausgabe bei jedem Spulenwechsel für eine bestimmte Datei ist möglich sowie auch die Ausgabe nach Verarbeitung einer bestimmten Anzahl von Sätzen einer Datei.

Format 1 der RERUN-Klausel (Auszug; vollständiges Format siehe [1]):



herstellername

Angabe SYSnnn (0 ≤ nnn ≤ 244)
COBOL85 erzeugt entweder eine Fixpunktdatei oder zwei Fixpunktdateien:

a) eine Fixpunktdatei, falls nnn ≤ 200.
COBOL85 bildet den Standardnamen `progid.RERUN.SYSnnn` sowie den Linknamen SYSnnn.
Die Fixpunkte werden fortlaufend in diese Datei geschrieben. Bei Dateiende wird intern weiterer Speicherbereich angefordert.

a) Zwei Fixpunktdateien, falls nnn > 200.
COBOL85 bildet die Standardnamen
`progid.RERUN.SYS.nnnA`,
`progid.RERUN.SYS.nnnB` und die Linknamen SYSnnnA und SYSnnnB.
Fixpunkte werden alternierend in beide Dateien ausgegeben, wobei ein zuvor geschriebener Fixpunkt überschrieben wird.

Das Format 2 der RERUN-Klausel ist nur für Sortierdateien möglich und wird deshalb im Abschnitt 10.3 beschrieben.
Nach jeder fehlerfreien Ausgabe eines Fixpunktes werden dem Benutzer auf SYSOUT Informationen für einen eventuellen Wiederanlauf gemeldet.
Format der Meldung :

```
EXC0301 CHECKPOINT 'aa' INITIATED ON HALF PAGE 'bb', DATE=cc, TIME=dd:ee
```

aa	Fixpunkt-Nummer
bb	PAM-Seiten-Nummer
cc	mm/tt/jj:Monat/Tag/Jahr
dd	Stunde
ee	Minute

11.2 Wiederanlauf

Mit dem RESTART-PROGRAM-Kommando startet der Benutzer ein ablauffähiges Programm bei einem durch einen Fixpunkt festgehaltenen Zustand.

Format des RESTART-PROGRAM-Kommandos (siehe [3]):

```
RESTART-PROGRAM dateiname,REST-OPT=START-PROG(CHECKPOINT=NUMBER(...)
```

dateiname	Name der Fixpunktdatei (COBOL-Standardname; siehe 11.1)
NUMBER(...)	Nummer der PAM-Seite, in der die Fixpunktsätze beginnen

Hinweise

1. Für den Wiederanlauf muß der Benutzer alle Betriebsmittel zuweisen, die vom wiederanlaufenden Programm benötigt werden, da bei Ausgabe des Fixpunktes Angaben über benötigte Betriebsmittel nicht sichergestellt werden.
2. Der Zustand der Benutzerdaten wird beim Wiederanlauf **nicht** automatisch wiederhergestellt. Also muß der Benutzer selbst seine Daten so wie zum Zeitpunkt der Fixpunktausgabe in geeigneter Weise zur Verfügung stellen.

12 Programmverknüpfungen

Ein Programmsystem besteht aus einem Hauptprogramm (das Programm, das auf Systemebene aufgerufen wird) und einem oder mehreren externen Unterprogrammen, die sowohl in der Sprache des Hauptprogramms als auch in anderen Programmiersprachen geschrieben sein können.

Den hierzu erforderlichen Programmverknüpfungen dienen die Inter-Language Communication Services (ILCS). ILCS ist Bestandteil des Common Runtime Environment (CRTE) und ist im CRTE-Benutzerhandbuch [2] beschrieben.

Hinweis

Abweichend von den ILCS-Konventionen ist bei der Übergabe mehrerer Parameter von einem COBOL-Programm an ein aufgerufenes Programm das höchstwertige Bit der letzten Parameteradreßkonstante der Liste gesetzt, es sei denn, einer oder mehrere Parameter der Liste werden „by value“ übergeben (siehe 12.3).

12.1 Binden und Laden von Unterprogrammen

Den Namen eines Unterprogramms kann man in der CALL-Anweisung entweder als Literal angeben oder als Bezeichner eines Datenfeldes, das den Unterprogrammnamen enthält. Abhängig von der Art des Unterprogrammaufrufs wird ein Programmsystem unterschiedlich gebunden und geladen.

Unterprogrammaufruf "CALL literal"

Der Name des Unterprogramms ist bereits zur Übersetzungszeit festgelegt. Der Compiler setzt auf diese Unterprogramme Externverweise ab, die in anschließenden Bindeläufen vom jeweils verwendeten Binder befriedigt werden. Enthält ein Programmsystem ausschließlich Aufrufe in der Form „CALL literal“, kann es, wie in Kapitel 6 beschrieben, zu einer permanent oder temporär ablauffähigen Programmausführungseinheit gebunden und anschließend geladen werden.

Unterprogrammaufruf "CALL bezeichner"

Der Name des Unterprogramms muß erst zum Ablaufzeitpunkt bekannt sein (z.B. nach Eingabe an der Datensichtstation). Für Unterprogramme, die nach Bedarf mit „CALL bezeichner“ aufgerufen werden, gibt es keine Externverweise; sie werden deshalb vom DBL während des Programmablaufs dynamisch nachgeladen. Programmsysteme mit derartigen Unterprogrammaufrufen können nur auf eine der folgenden Arten zum Ablauf gebracht werden:

1. Mit dem DBL die bei der Übersetzung entstandenen Module dynamisch binden und die Unterprogramme ohne Externverweise dynamisch nachladen.
2. Mit dem TSOSLNK ein Großmodul vorbinden, das das Hauptprogramm sowie die Unterprogramme mit Externverweisen enthält. Mit dem DBL das Großmodul aufrufen und die Unterprogramme ohne Externverweise dynamisch nachladen.
3. Mit dem BINDER einen LLM oder mehrere LLMs (vor)binden. Mit dem Bindelader den (Groß-)LLM bzw. den LLM, der das Hauptprogramm enthält, aufrufen und die Unterprogramme ohne Externverweise dynamisch nachladen.

Vor dem Aufruf des Bindeladers müssen im Regelfall folgende Zuweisungen vorgenommen werden:

```
/SET-FILE-LINK COBOBJCT,bibliothek
```

für die Bibliothek, in der die nachzuladenden Unterprogramme stehen,
sowie

```
/SET-FILE-LINK BLSLIBnn,laufzeitbibliothek
```

für das Common Run-Time Environment (CRTE), das das COBOL-Laufzeitsystem enthält.

Dabei ist zu beachten, daß die Verwendung des Linknamens BLSLIBnn nur möglich ist, wenn im Aufrufkommando für den DBL

```
RUN-MODE=ADVANCED(ALTERNATE-LIBRARIES=YES)
```

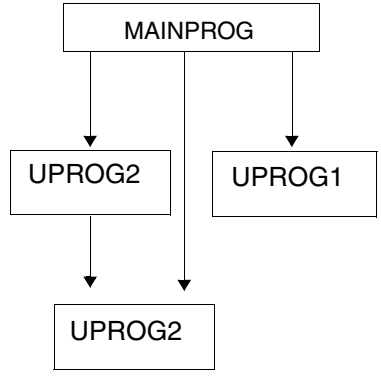
angegeben wird (siehe Abschnitt 6.4).

Hinweis

In Bindelademodulen (LLMs) können in CALL bezeichner- bzw. CANCEL bezeichner-Anweisungen Programmnamen angegeben werden, die bis zu 30 Zeichen lang sind. In Objektmodulen dürfen die Programmnamen nicht länger als acht Zeichen lang sein. Bei CANCEL bezeichner-Anweisungen müssen diese Namen in der *run-unit* in den ersten sieben Zeichen eindeutig sein und das achte Zeichen darf kein Bindestrich '-' sein.

Beispiel 12-1: Binde- und Ladetechniken für Programmsysteme mit dynamisch nachzuladenden Unterprogrammen

Programmkonstellation und Art der Aufrufe



MAINPROG:

```
...
CALL "UPROG1" USING ...
MOVE "UPROG2" TO bezeichner-1
MOVE "UPROG3" TO bezeichner-2
CALL bezeichner-1 USING bezeichner-2
...
```

UPROG2:

```
    ...
    PROCEDURE DIVISION USING bezeichner-2
    CALL bezeichner-2
```

```
...
CALL "UPROG3"
...
```

UPROG1 wird ausschließlich in der Form "CALL literal" aufgerufen.
UPROG2 wird ausschließlich in der Form "CALL bezeichner" aufgerufen.
UPROG3 wird auf beide Arten aufgerufen.

Das bedeutet: Für UPROG1 und UPROG3 werden Externverweise abgesetzt, UPROG2 wird dynamisch nachgeladen.

Für diese Programmkonstellation werden im folgenden die Möglichkeiten gezeigt, das Programm zum Ablauf zu bringen.

Die einzelnen Programme sind als Objektmodule unter den Elementnamen MAINPROG, UPROG1, UPROG2 und UPROG3 in der Bibliothek BENUTZER-PROGRAMME abgelegt.

1. Verwendung des DBL (dynamisches Binden)

```
/SET-FILE-LINK BLSLIB00,$.SYSLNK.CRTE   ———————————————————————— (1)
/SET-FILE-LINK COBOBJCT,BENUTZER-PROGRAMME   ———————————————————— (2)
/START-PROGRAM *MODULE(LIB=BENUTZER-PROGRAMME,ELEM=MAINPROG,-    ———— (3)
 RUN-MODE=ADVANCED(ALT-LIB=YES,UNRES-EXT=DELAY,-
 LOAD-INF=REFERENCES))
```

(1) Zuweisung der Laufzeitbibliothek

(2) Zuweisung der Bibliothek, aus der das Unterprogramm UPROG2 dynamisch nachgeladen wird

(3) Aufruf des Objektmoduls mit dem Hauptprogramm MAINPROG. Aus der hier angegebenen Bibliothek BENUTZER-PROGRAMME befriedigt der Bindelader die Externverweise auf die Unterprogramme UPROG1 und UPROG3. Die Operanden UNRESOLVED-EXTERNAL=DELAY und LOAD-INFORMATION=REFERENCES müssen immer dann angegeben werden, wenn die Ladeeinheit noch offene bedingte Externverweise (WXTRNs) enthält (siehe [9]). Dies ist z.B. dann der Fall, wenn im Unterprogramm weitere Dateien mit anderer Dateiorganisation als im Hauptprogramm verarbeitet werden sollen.

2. Verwendung des TSOSLNK (Großmodulbinden)

```
/START-PROGRAM $TSOSLNK
 MODULE GROSSMOD,LET=Y,LIB=MODUL.LIB ————————————————————————— (1)
 INCLUDE MAINPROG,BENUTZER-PROGRAMME ————————————————————————— (2)
 RESOLVE ,BENUTZER-PROGRAMME  ———————————————————————————————— (3)
 RESOLVE ,$.SYSLNK.CRTE  ————————————————————————————————————— (4)
 LINK-SYMBOLS *KEEP ——————————————————————————————————————— (5)
 END

/SET-FILE-LINK BLSLIB00,$.SYSLNK.CRTE ——————————————————————— (6)
/SET-FILE-LINK COBOBJCT,BENUTZER-PROGRAMME  ————————————————— (7)
/START-PROGRAM *MODULE(LIB=MODUL.LIB,ELEM=GROSSMOD,-  ——————— (8)
 RUN-MODE=ADVANCED(ALT-LIB=YES,UNRES-EXT=DELAY,-
 LOAD-INFO=REFERENCE))
```

(1) Das zu erstellende Großmodul GROSSMOD wird in der Bibliothek MODUL.LIB abgelegt.

(2) Einbinden des Moduls MAINPROG aus der Bibliothek BENUTZER-PROGRAMME

(3) Einbinden der Bibliothek BENUTZER-PROGRAMME zur Befriedigung der Externverweise auf UPROG1 und UPROG3

(4) Einbinden der Bibliothek, die die Laufzeitroutinen enthält

(5) Mit dieser Anweisung werden die Symbole für die Einsprungstellen und die Programmabschnitte für den späteren Ablauf mit dem Bindelader sichtbar gehalten.

(6) Zuweisung der Laufzeitbibliothek

(7) Zuweisung der Bibliothek, aus der das Unterprogramm UPROG2 dynamisch nachgeladen wird

(8) Aufruf des Großmoduls GROSSMOD. Die Operanden UNRESOLVED-EXTERNAL=DELAY und LOAD-INFORMATION=REFERENCES müssen immer dann angegeben werden, wenn die Ladeeinheit noch offene bedingte Externverweise (WXTRNs) enthält (siehe [9]). Dies ist z.B. dann der Fall, wenn im Unterprogramm weitere Dateien mit anderer Dateiorganisation als im Hauptprogramm verarbeitet werden sollen.

3. Verwendung des BINDER (LLM-Binden)

Im Unterschied zum TSOSLNK läßt der BINDER standardmäßig alle Externverweise und Einsprungpunkte sichtbar; dies ist für den anschließenden Bindelader-Lauf unbedingt erforderlich.
Ferner können bei Verwendung des BINDER die Externverweise offen bleiben; deshalb braucht das LZS nicht eingebunden zu werden. Dies ist von Vorteil, wenn für den Programmablauf ein gemeinsam benutzbares LZS verwendet werden soll.

a) Erzeugen eines einzigen Bindelademoduls

```
/START-PROGRAM $BINDER
 START-LLM-CREA GROSSMOD  ------------------------------------------------- (1)
 INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,ELEM=MAINPROG ---------------------- (2)
 INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,ELEM=UPROG2 ------------------------ (3)
 RESOLVE-BY-AUTOLINK LIB=BENUTZER-PROGRAMME ------------------------------- (4)
 SAVE-LLM LIB=MODUL.LIB  -------------------------------------------------- (5)
 END
/SET-FILE-LINK BLSLIB00,$.SYSLNK.CRTE  ------------------------------------ (6)
/START-PROGRAM *MODULE(LIB=MODUL.LIB,ELEM=GROSSMOD,-   -------------------- (7)
 RUN-MODE=ADVANCED(ALT-LIB=YES,UNRES-EXT=DELAY,-
 LOAD-INFO=REFERENCE))
```

(1) Erzeugen eines Bindelademoduls mit dem Namen GROSSMOD.

(2) Explizites Einbinden des Hauptprogramm-Moduls MAINPROG aus der Bibliothek BENUTZER-PROGRAMME

(3) Explizites Einbinden des Moduls UPROG2 aus der Bibliothek BENUTZER-PROGRAMME, um dynamisches Nachladen zu vermeiden; damit erübrigt sich beim anschließenden Bindeladevorgang die Zuweisung der Bibliothek BENUTZER-PROGRAMME mit dem Linknamen COBOBJCT.

(4) Einbinden aller weiteren erforderlichen Module (UPROG1, UPROG3) aus der Bibliothek BENUTZER-PROGRAMME

(5) Abspeichern des erzeugten Bindelademoduls in der Programmbibliothek MODUL.LIB als Element vom Typ L

(6) Zuweisen der Laufzeitbibliothek

(7) Aufruf des Bindelademoduls GROSSMOD. Die Operanden UNRESOLVED-EXTERNAL=DELAY und LOAD-INFORMATION=REFERENCES müssen immer dann angegeben werden, wenn die Ladeeinheit noch offene bedingte Externverweise (WXTRNs) enthält (siehe [9]). Dies ist z.B. dann der Fall, wenn im Unterprogramm weitere Dateien mit anderer Dateiorganisation als im Hauptprogramm verarbeitet werden sollen.

b) Umwandeln der Objektmodule in einzelne Bindelademodule

```
/START-PROGRAM $BINDER
 ...
 START-LLM-CREATION INTERNAL-NAME=MAINPROG                    )
 INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,ELEM=MAINPROG         }———————————— (1)
 SAVE-LLM LIB=MODULE.LLM                                      )
 ...
 START-LLM-CREATION INTERNAL-NAME=UPROG1                      )
 INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,ELEM=UPROG1           }———————————— (2)
 SAVE-LLM LIB=MODULE.LLM,ENTRY-POINT=UPROG1                   )
 ...
 START-LLM-CREATION INTERNAL-NAME=UNTER2                      )
 INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,ELEM=UPROG2           }———————————— (3)
 SAVE-LLM LIB=MODULE.LLM,ENTRY-POINT=UPROG2                   )
 ...
 START-LLM-CREATION INTERNAL-NAME=UPROG3                      )
 INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,ELEM=UPROG3           }———————————— (4)
 SAVE-LLM LIB=MODULE.LLM                                      )
 END
 ...
/SET-FILE-LINK BLSLIB00,$.SYSLNK.CRTE————————————————————————————————— (5)
/SET-FILE-LINK COBOBJCT,MODULE.LLM————————————————————————————————————— (6)
/START-PROGRAM *MODULE(LIB=MODULE.LLM,ELEM=MAINPROG,-  ——————————————————— (7)
 RUN-MODE=ADVANCED(ALT-LIB=YES,UNRES-EXT=DELAY,-
 LOAD-INFO=REFERENCE))
```

(1) Erzeugen eines LLM namens MAINPROG; der Name des LLM ist frei wählbar. Einbinden des Objektmoduls MAINPROG aus der Bibliothek BENUTZER-PROGRAMME. Da im SAVE-LLM-Kommando der Elementname weggelassen ist, gilt als Elementname die Angabe aus dem START-LLM-CREATION-Kommando, nämlich MAINPROG.

(2) Erzeugen eines LLM namens UPROG1. Einbinden des Objektmoduls UPROG1 aus der Bibliothek BENUTZER-PROGRAMME. Mit ENTRY-POINT=UPROG1 ist dieser LLM als Unterprogramm definiert.

(3) Erzeugen eines LLM namens UNTER2. Einbinden des Objektmoduls UPROG2 aus der Bibliothek BENUTZER-PROGRAMME. Mit ENTRY-POINT=UPROG2 ist dieser LLM als Unterprogramm definiert.

(4) Erzeugen eines LLM namens UPROG3. Einbinden des Objektmoduls UPROG3 aus der Bibliothek BENUTZER-PROGRAMME. Da im SAVE-LLM-Kommando kein ENTRY-POINT angegeben ist, kann UPROG3 sowohl als Unterprogramm als auch als Hauptprogramm verwendet werden.

(5) Zuweisen der Laufzeitbibliothek

(6) Zuweisen der Bibliothek, in der die LLMs stehen, mit dem Linknamen COBOBJCT, damit die offenen Externverweise der zuvor erzeugten LLMs befriedigt werden können.

(7) Aufruf des LLM mit dem Hauptprogramm MAINPROG. Die Operanden UNRESOLVED-EXTERNAL=DELAY und LOAD-INFORMATION=REFERENCES müssen immer dann angegeben werden, wenn die Ladeeinheit noch offene bedingte Externverweise (WXTRNs) enthält (siehe [9]). Dies ist z.B. dann der Fall, wenn im Unterprogramm weitere Dateien mit anderer Dateiorganisation als im Hauptprogramm verarbeitet werden sollen.

12.2 COBOL-Sonderregister RETURN-CODE

Das COBOL-Sonderregister RETURN-CODE kann zur Verständigung zwischen getrennt übersetzten COBOL-Programmen einer Ablaufeinheit dienen. Das Sonderregister existiert nur einmal im Programmsystem und ist intern als neunstelliges binäres Datenfeld (PIC S9(9) COMP-5 SYNC) definiert.
RETURN-CODE kann während des Ablaufs beliebig von den einzelnen getrennt übersetzten Programmen abgefragt oder verändert werden. Bei Beendigung des Programmlaufs wird vom Laufzeitsystem überprüft, ob RETURN-CODE auf Null steht. Ist das nicht der Fall, wird die Fehlermeldung COB9119 (bei COBOL-Returncode > 0) bzw. COB9128 (bei Anwender-Returncode > 0) ausgegeben. Wurde das Programm innerhalb einer Prozedur aufgerufen, verzweigt das Programm zum nächsten STEP- , ABEND-, ABORT-, ENDP- oder LOGOFF-Kommando.

Ferner wird beim Verlassen eines COBOL-Unterprogramms der Wert des Sonderregisters RETURN-CODE in die Register 0 und 1 geladen. Entsprechend den ILCS-Konventionen steht der Wert damit dem aufrufenden Programm als Funktionswert zur Verfügung.

Um einen Funktionswert aus einem C-Programm zu übernehmen, muß das aufrufende COBOL-Programm mit der Steueranweisung
RETURN-CODE=FROM-ALL-SUBPROGRAMS der RUNTIME-OPTIONS-Option
bzw. mit dem COMOPT-Operanden ACTIVATE-XPG4-RETURNCODE=YES übersetzt werden.

Um die abnormale Beendigung des Programms zu vermeiden, muß der Benutzer dafür sorgen, daß RETURN-CODE vor Erreichen der STOP RUN-Anweisung den Wert 0 enthält.

12.3 Parameterübergabe an C-Programme

Um eine C-konforme, direkte Wertübergabe an C-Programme zu ermöglichen, existiert in beiden Sprachformaten der CALL-Anweisung die Angabe

```
USING BY VALUE {bezeichner-3} ...
```

bezeichner-3 muß als 2 oder 4 Byte langes Datenfeld mit USAGE COMP-5 oder als 1 Byte langes Datenfeld definiert sein. Andernfalls ist das Ergebnis der Parameterübergabe unbestimmt.
Die Parameterübergabe „by value“ bewirkt, daß nur der Wert des Parameters an das aufgerufene C-Programm übergeben wird. Das aufgerufene Programm kann auf diesen Wert zugreifen und ihn verändern, wobei der Wert des Parameters im COBOL-Programm unverändert bleibt.
Wird in einer Liste von Parametern mindestens einer „by value“ übergeben, so ist in der letzten Parameteradreßkonstanten der Liste das höchstwertige Bit nicht gesetzt..

13 COBOL85 und POSIX

In COBOL85-BC nicht unterstützt !

Der Compiler COBOL85 V2.3 kann in der POSIX-Umgebung (POSIX-Shell) aufgerufen und mit Optionen gesteuert werden.

Ferner können COBOL-Programme, die in POSIX oder BS2000 übersetzt wurden, in der POSIX-Umgebung zum Ablauf gebracht werden.

Schließlich kann, falls das POSIX-Subsystem vorhanden ist, auch bei Compiler- bzw. Programmablauf im BS2000 auf das POSIX-Dateisystem zugegriffen werden.

Auf folgende weiterführende Literatur zum Thema POSIX sei an dieser Stelle hingewiesen:

POSIX im BS2000/OSD

Diese Broschüre gibt einen allgemeinen Überblick über die Strategien und Ziele von POSIX im BS2000/OSD.

POSIX - Grundlagen

Dieses Handbuch bietet eine anschauliche Einführung in POSIX und vermittelt alle Grundkenntnisse, die für das Arbeiten mit dem POSIX-Subsystem benötigt werden.

POSIX - Kommandos

Dieses Handbuch enthält die Beschreibung aller POSIX-Benutzerkommandos.

13.1 Überblick

Die folgenden drei Abschnitte bieten einen Überblick über den Einsatz des Compilers im POSIX-Subsystem.

13.1.1 Übersetzen

Für das Übersetzen von COBOL-Quellprogrammen steht das POSIX-Kommando `cobol` zur Verfügung. Dieses Kommando ist Abschnitt 13.3, S.282, ausführlich beschrieben.

Erzeugen einer LLM-Objektdatei („.o“-Datei)

Der Compiler erzeugt pro übersetzter Quelldatei ein LLM und legt dieses im aktuellen Dateiverzeichnis als POSIX-Objektdatei mit dem Standardnamen *basisname*.o ab. *basisname* ist der Name der Quelldatei ohne die Dateiverzeichnisbestandteile und ohne das Suffix .cob oder .cbl.
Bei der Übersetzung von Quellprogramm-Folgen wird für jedes Quellprogramm ein LLM erzeugt, das in einer POSIX-Objektdatei abgelegt wird. *basisname* ist in diesem Fall für das zweite bis letzte Quellprogramm der jeweilige PROGRAM-ID-Name, wobei Kleinbuchstaben gegebenenfalls in Großbuchstaben umgesetzt werden.
Standardmäßig wird nach dem Übersetzungslauf ein Bindelauf gestartet.
Mit der Option `-c` kann der Bindelauf verhindert werden (siehe S.283).

Erzeugen einer Übersetzungsliste

Mit der Option `-P` (siehe S.286) können diverse Übersetzungslisten angefordert werden (z.B. Quellprogrammliste, Fehlerliste, Querverweisliste etc.). Die angeforderten Listen schreibt der Compiler in eine Listendatei mit dem Standardnamen *basisname*.lst und legt diese im aktuellen Dateiverzeichnis ab. *basisname* ist der Name der Quelldatei ohne die Dateiverzeichnisbestandteile und ohne das Suffix .cob oder .cbl. In solch einem Fall kann der Name der Quelldatei auch mit der Option *-k dateiname* angegeben werden.

Für das Ausdrucken von Listendateien steht das POSIX-Kommando `lp` zur Verfügung (siehe Handbuch „POSIX-Kommandos“).

Beispiel für das Ausdrucken einer Übersetzungsliste

```
lp -o control-mode=*physical cobbsp.lst
```

Ausgabeziele und Ausgabe-Code

Der Compiler legt die Ausgabedateien im aktuellen Dateiverzeichnis ab, d.h. in dem Dateiverzeichnis, aus dem der Compilerlauf gestartet wird.
Zeichen- und Zeichenketten-Konstanten im Programm (Objektdatei) werden immer im EBCDI-Code abgelegt
Sofern von der Möglichkeit Gebrauch gemacht wird, das POSIX Dateisystem auf einem gemounteten UNIX Dateisystem abzulegen bzw. mit UNIX Werkzeugen die POSIX Dateien im ASCII Code zu bearbeiten, sind Fehlerdateien (ERRFIL oder umgelenkte Bildschirmausgaben) in jedem Fall außerhalb des POSIX Dateisystems im BS2000 abzulegen, da für solche Dateien eine Code-Konversion nur eingeschränkt zur Verfügung steht.

13.1.2 Binden

Ein COBOL-Programm wird in der POSIX-Shell mit dem Aufrufkommando `cobol` zu einer ausführbaren Datei gebunden.

Ein Bindelauf wird automatisch gestartet, wenn die Option `-c` nicht angegeben wird (siehe S.283) und wenn bei einer ggf. vorangegangenen Übersetzung kein schwerwiegender Fehler auftrat.
Das fertig gebundene Programm wird als LLM in eine ausführbare POSIX-Datei geschrieben. Der Name dieser Datei sowie das Dateiverzeichnis werden mit der Binder-Option `-o` festgelegt. Ohne Angabe dieser Option wird die ausführbare POSIX-Datei unter dem Standardnamen a.out im aktuellen Dateiverzeichnis abgelegt.

Beim Binden in der POSIX-Shell können keine Binder-Listen erzeugt werden. Im Fehlerfall werden entsprechende Fehlermeldungen auf stderr ausgegeben.

Binden von Benutzermodulen

Benutzereigene Module können statisch und dynamisch (d.h. zum Ablaufzeitpunkt) eingebunden werden. Programme, die „unresolved externals“ auf Benutzermodule enthalten, können in der POSIX-Shell nicht gestartet werden.

Eingabequellen für den Binder können sein:

- vom Compiler erzeugte Objektdateien („.o“-Dateien)
- mit dem Dienstprogramm `ar` erstellte Bibliotheken („.a“-Dateien)
- LLMs, die mit dem POSIX-Kommando `bs2cp` aus PLAM-Bibliotheken in POSIX-Objektdateien kopiert wurden. Dies können LLMs sein, die in BS2000-Umgebung direkt von einem Compiler erzeugt wurden, oder Objektmodule, die mit dem BINDER in ein LLM geschrieben wurden.

– LLMs und Objektmodule, die in BS2000-PLAM-Bibliotheken stehen. Die PLAM-Bibliotheken müssen dazu mit den Umgebungsvariablen BLSLIB*nn* zugewiesen werden (siehe Operand `-l BLSLIB`, S.288).

Die Module können von jedem ILCS-fähigen BS2000-Compiler erzeugte Module sein (z.B. COBOL85, C, C++, ASSEMBH, FORTRAN90).

Wenn vom COBOL85-Compiler in BS2000-Umgebung erzeugte Module eingebunden werden sollen, müssen diese mit der Option ENABLE-UFS-ACCESS=YES übersetzt worden sein.

Für POSIX-Objektdateien werden beim Bindelauf intern INCLUDE-MODULES-Anweisungen abgesetzt, für ar-Bibliotheken und PLAM-Bibliotheken RESOLVE-BY-AUTOLINK-Anweisungen. Die Module werden in der nachfolgend beschriebenen Reihenfolge eingebunden.

Beim Binden ist mit der Option `-M` der Name des COBOL-Hauptprogramms (PROGRAM-ID-Name) anzugeben. Ohne diese Angabe nimmt der Binder an, daß das Hauptprogramm das C-Programm main() ist.

Binden der CRTE-Laufzeitbibliotheken

Die offenen Externbezüge auf das COBOL85-Laufzeitsystem werden vom Binder automatisch aus der CRTE-Bibliothek SYSLNK.CRTE.PARTIAL-BIND aufgelöst.

Binde-Reihenfolge

1. Alle explizit angegebenen Objektdateien („.o“-Dateien) mit INCLUDE-MODULES-Anweisungen und ggf. alle explizit angegebenen ar-Bibliotheken („.a“-Dateien). Für jede ar-Bibliothek wird eine eigene RESOLVE-BY-AUTOLINK-Anweisung abgesetzt.
2. Alle vom Compiler bei der Übersetzung generierten Objektdateien mit INCLUDE-MODULES-Anweisungen
3. Alle mit den Optionen `-l` und `-L` angegebenen ar-Bibliotheken sowie die mit `-l BLSLIB` zugewiesenen PLAM-Bibliotheken. Für jede ar-Bibliothek wird eine eigene RESOLVE-BY-AUTOLINK-Anweisung abgesetzt. Die mit `-l BLSLIB` zugewiesenen PLAM-Bibliotheken werden in einer einzigen RESOLVE-BY-AUTOLINK in einer Liste an den BINDER übergeben.
4. Die CRTE-Bibliothek (SYSLNK.CRTE.PARTIAL-BIND) und ggf. die SORT-Bibliothek (SORTLIB)

Die in den Schritten 1. bis 3. bearbeiteten Objektdateien und Bibliotheken werden jeweils in der Reihenfolge eingebunden, in der sie in der Kommandozeile angegeben werden. Bei den vom Compiler generierten Objektdateien (siehe 2.) richtet sich die Bindereihenfolge nach der Reihenfolge der zugehörigen Quelldateien.

Beispiel

```
export BLSLIB99=‘$MYTEST.LIB2‘
export BLSLIB01=‘$MYTEST.LIB1‘
cobol -M COBBSP -o cobbsp cobupro1.cob cobupro2.cob cobbsp.o cobupro3-5.a
-L /usr/private -l xyz -l BLSLIB
```

Bindereihenfolge:

1. cobbsp.o
2. cobupro3-5.a
3. cobupro1.o
4. cobupro2.o
5. /usr/private/libxyz.a
6. $MYTEST.LIB1
7. $MYTEST.LIB2
8. Laufzeitbibliotheken

13.1.3 Testen

Fertig gebundene Programme können mit der Dialogtesthilfe AID getestet werden. Voraussetzung hierfür sind Testhilfeinformationen (LSD), die der Compiler bei Angabe der Option `-g` (siehe S.289) erzeugt.

Die Testhilfe AID wird von einem BS2000-Terminal aus mit dem POSIX-Kommando `debug` *programmname [argumente]* aktiviert.
Nach Eingabe dieses Kommandos ist die BS2000-Umgebung die aktuelle Umgebung. Dies wird mit dem Prompting %DEBUG/ angezeigt. In diesem Modus können die Testhilfe-Kommandos so eingegeben werden, wie im Handbuch „AID Testen von COBOL-Programmen“ [8] beschrieben. Nach Beendigung des Programms ist wieder die POSIX-Shell die aktuelle Umgebung.

Das `debug`-Kommando ist im Handbuch „POSIX-Kommandos“ [31] beschrieben.

13.2 Bereitstellen des Quellprogramms

Der COBOL85-Compiler erkennt COBOL-Quelldateien an einem der folgenden Standard-Suffixe:

`.cob` oder `.cbl`

COBOL-Quelldateien, deren Dateinamen nicht mit einem Standard-Suffix enden, können ebenfalls übersetzt werden, wenn die Dateinamen mit der Option **-k** angegeben werden (siehe S.283).

Quellprogramme, die in BS2000-Dateien oder PLAM-Bibliotheken abgelegt sind, können mit dem Compiler im POSIX-Subsystem nicht verarbeitet werden.

Für das Transferieren von BS2000-Dateien und PLAM-Bibliothekselementen in das POSIX-Dateisystem und umgekehrt steht das POSIX-Kommando `bs2cp` zur Verfügung.

Für das Editieren von POSIX-Dateien von einem BS2000-Terminal aus steht das POSIX-Kommando `edt` zur Verfügung.
Erfolgte der Zugang zum POSIX-Subsystem von einem SINIX-Terminal aus, steht zum Editieren das POSIX-Kommando `vi` zur Verfügung (siehe Handbuch „POSIX-Kommandos“ [31]).

Eingabe von Quellprogrammteilen (COPY-Elemente)

Für das Kopieren von COPY-Texten aus POSIX-Dateien wird die COPY-Anweisung wie folgt ausgewertet:

```
COPY textname [IN/OF bibliotheksname]
```

textname ist der Name der POSIX-Datei (ohne Dateiverzeichnisbestandteile), die den COPY-Text enthält. Der Name darf keine Kleinbuchstaben enthalten.

bibliotheksname ist der Name einer Umgebungsvariablen, die einen oder mehrere absolute Pfadnamen der zu durchsuchenden Dateiverzeichnisse enthält. Der Name darf keine Kleinbuchstaben enthalten.

Fehlt die Angabe IN/OF *bibliotheksname* in der COPY-Anweisung, wertet der Compiler den Inhalt einer Umgebungsvariablen namens COBLIB aus.

Die Umgebungsvariablen müssen vor Aufruf des Compilers mit den Pfadnamen der zu durchsuchenden Dateiverzeichnisse versorgt und mit dem POSIX-Kommando `export` exportiert werden.

Beispiel

COPY-Anweisungen im Quellprogramm:

```
...
COPY TEXT1 IN COPYLNK
COPY TEXT2 IN COPYLNK
...
```

Definieren und Exportieren der Umgebungsvariablen in der POSIX-Shell:

```
export COPYLNK=/USERIDXY/copy1:/USERIDXY/copy2
```

Dadurch wird die Umgebungsvariable COPYLNK mit den durch Doppelpunkt getrennten Namen von zwei Dateiverzeichnissen initialisiert, die nach den POSIX-Dateien mit den COPY-Texten (TEXT1, TEXT2) durchsucht werden sollen. Zuerst wird das Verzeichnis /USERIDXY/copy1, anschließend das Verzeichnis /USERIDXY/copy2 durchsucht.

13.3 Steuerung des Compilers

Der COBOL85-Compiler kann in der POSIX-Shell mit dem Kommando **cobol** aufgerufen und mit Optionen gesteuert werden.

Aufruf-Syntax

`cobol␣`*option*`␣ ... `*eingabedatei*`␣ ...`

Eingaberegeln

1. Optionen und Eingabedateien können in der Kommandozeile gemischt angegeben werden.
2. Optionen ohne Argumente (z.B. `-c`, `-v`, `-g`) dürfen gruppiert werden (z.B. `-cvg`).
3. Unzulässig ist dagegen die Gruppierung der diversen Argumente einer Option (z.B. von `-C`). Option und Argument müssen durch ein Leerzeichen (␣) getrennt werden (z.B. `-C EXPAND-COPY=YES`).
4. Folgende Optionen können mehrfach in der Kommandozeile vorkommen:

   `-C, -CC, -k, -L, -P, -l`

   Alle anderen Optionen dürfen nur einmal verwendet werden. Wenn dennoch mehr als eine dieser Optionen angegeben wird, gilt die jeweils letzte Option in der Kommandozeile.

Standardmäßig, d.h. wenn nicht mit der Option `-c` der Compilerlauf nach der Übersetzung beendet wird und wenn die Übersetzung ohne schwerwiegenden Fehler verlaufen ist, wird automatisch ein Bindelauf gestartet.

Falls ein C-Compiler vorhanden ist, können mit dem Kommando cobol neben COBOL-Quellprogrammen gleichzeitig auch C-Quellprogramme übersetzt und zu der COBOL-Anwendung gebunden werden.

Die Optionen zur Steuerung des Übersetzungs- und Bindelaufs sind nachfolgend beschrieben.

13.3.1 Allgemeine Optionen

–c

Der Compilerlauf wird beendet, nachdem für jede übersetzte Quelldatei ein LLM erzeugt und in eine Objektdatei *basisname*.o abgelegt wurde. *basisname* ist der Name der Quelldatei ohne die Dateiverzeichnisbestandteile und ohne das Suffix `.cbl` oder `.cob`. Die Objektdatei wird in das aktuelle Dateiverzeichnis geschrieben.

Wenn ein Quellprogramm ohne Angabe dieser Option übersetzt wird, wird nach der Übersetzung ein Bindelauf gestartet.

–k *dateiname*

Mit dieser Option kann eine COBOL-Quelldatei angegeben werden, deren Dateiname nicht mit dem Suffix `.cbl` oder `.cob` endet.

Wenn der mit `-k` angegebene Quelldateiname dennoch mit dem Suffix `.cbl` oder `.cob` endet, wird dieses Suffix bei der Bildung des Basisnamens für die Objekt- und Listendateien mit dem Suffix `.o` bzw. `.lst` überschrieben.

–v

Bei Angabe dieser Option werden folgende Informationen auf dem Bildschirm ausgegeben:

- Copyright und Versionsangabe des COBOL85-Compilers und des cobol-Kommandos
- Meldungen des COBOL85-Compilers über akzeptierte Steueranweisungen
- Summe aller Hinweis- und Fehlermeldungen des Übersetzungslaufs
- verbrauchte CPU-Zeit
- die vollständige Kommandozeile für den Binderaufruf

Diese Option betrifft nur die Ausgaben des COBOL85-Compilers.

Wenn bei der gleichzeitigen Übersetzung von C-Quellen entsprechende Meldungen des C-Compilers ausgegeben werden sollen, muß die Option `-CC -V` verwendet werden.

–W *err-level*

Diese Option wird intern auf COMOPT MINIMAL-SEVERITY = err-level abgebildet. Die COMOPT MINIMAL-SEVERITY sollte deshalb nicht mit -C übergeben werden.
In der Fehlerliste stehen keine Meldungen, deren Fehlergewicht kleiner ist als der angegebene Wert. Für *err-level* sind folgende Angaben möglich:

I	Information (Voreinstellung)
0	Warnung
1	Fehler
2	Schwerwiegender Fehler
3	Abbruchfehler

13.3.2 Option für Compiler-Anweisungen

–C *comopt*

Für *comopt* können alle in der folgenden Übersicht aufgeführten COMOPT-Anweisungen in Voll- oder Abkürzungsschreibweise angegeben werden.
Die Wirkungsweise der einzelnen COMOPTs ist in Kapitel 4 beschrieben.

Beispiel

`-C SET-FUNCTION-ERROR-DEFAULT=YES` oder `-C S-F-E-D=YES`

Übersicht: COMOPTs, die mit der Option –C übergeben werden können

COMOPT	mögliche Abkürzung
ACCEPT-LOW-TO-UP={YES/NO}	ACC-L-T-U
ACTIVATE-WARNING-MECHANISM={YES/NO}	ACT-W-MECH
ACTIVATE-XPG4-RETURNCODE={YES/NO}	
CHECK-CALLING-HIERARCHY={YES/NO}	CHECK-C-H
CHECK-DATE={YES/NO}	CHECK-D
CHECK-FUNCTION-ARGUMENTS={YES/NO}	CHECK-FUNC
CHECK-PARAMETER-COUNT={YES/NO}	CHECK-PAR-C
CHECK-REFERENCE-MODIFICATION={YES/NO}	CHECK-REF
CHECK-SCOPE-TERMINATORS={YES/NO}	CHECK-S-T
CHECK-SOURCE-SEQUENCE={YES/NO}	CHECK-S-SEQ
CHECK-TABLE-ACCESS={YES/NO}	CHECK-TAB

CONTINUE-AFTER-MESSAGE={YES/NO}	CON-A-MESS
EXPAND-COPY={YES/NO}	EXP-COPY
FLAG-ABOVE-INTERMEDIATE={YES/NO}	
FLAG-ABOVE-MINIMUM={YES/NO}	
FLAG-ALL-SEGMENTATION={YES/NO}	
FLAG-INTRINSIC-FUNCTIONS={YES/NO}	
FLAG-NONSTANDARD={YES/NO}	
FLAG-OBSOLETE={YES/NO}	
FLAG-REPORT-WRITER={YES/NO}	
FLAG-SEGMENTATION-ABOVE1={YES/NO}	
GENERATE-INITIAL-STATE={YES/NO}	GEN-INIT-STA
GENERATE-LINE-NUMBER={YES/NO}	GEN-L-NUM
GENERATE-SHARED-CODE={YES/NO}	GEN-SHARE
IGNORE-COPY-SUPPRESS={YES/NO}	IGN-C-SUP
INHIBIT-BAD-SIGN-PROPAGATION={YES/NO}	
LINE-LENGTH=132 / 119..172	LINE-L
LINES-PER-PAGE=64 / 20..128	LINES
MARK-NEW-KEYWORDS={YES/NO}	M-N-K
MAXIMUM-ERROR-NUMBER=ganzzahl	MAX-ERR
MERGE-DIAGNOSTICS={YES/NO}	M-DIAG
OPTIMIZE-CALL-IDENTIFIER={YES/NO}	O-C-I
RESET-PERFORM-EXITS={YES/NO}	RES-PERF
ROUND-FLOAT-RESULTS-DECIMAL={YES/NO}	ROUND-FLOAT
SEPARATE-TESTPOINTS={YES/NO}	SEP-TESTP
SET-FUNCTION-ERROR-DEFAULT={YES/NO}	S-F-E-D
SHORTEN-OBJECT={YES/NO}	SHORT-OBJ
SHORTEN-XREF={YES/NO}	SHORT-XREF
SORT-EBCDIC-DIN={YES/NO}	SORT-E-D
SORT-MAP={YES/NO}	
SUPPORT-WINDOW-DEBUGGING={YES/NO}	S-W-D
SUPPRESS-LISTINGS={YES/NO}	SUP-LIST
SUPPRESS-MODULE={YES/NO}	SUP-MOD
TCBENTRY=name	
TERMINATE-AFTER-SEMANTIC={YES/NO}	TERM-A-SEM

TERMINATE-AFTER-SYNTAX={YES/NO}	TERM-A-SYN
TEST-WITH-COLUMN1={YES/NO}	TEST-W-C
USE-APOSTROPHE={YES/NO}	USE-AP

13.3.3 Option zur Ausgabe von Übersetzungsprotokollen

–P "(*listenangabe, ...*)"

Mit dieser Option wird gesteuert, welche Übersetzungsprotokolle vom Compiler erzeugt werden.
Diese Option wird intern auf COMOPT SYSLIST=(listenangabe,...) abgebildet. Die COMOPT SYSLIST sollte deshalb nicht mit -C übergeben werden.
Mit *listenangabe* können (analog zu COMOPT SYSLIST im BS2000) folgende Werte in einer Liste angegeben werden:

```
OPTIONS
NOOPTIONS
SOURCE
NOSOURCE
MAP
NOMAP
DIAG
NODIAG
XREF
NOXREF
ALL
NO
```

Standardmäßig (NO) werden keine Übersetzungsprotokolle erzeugt.

Die mit `-P` angeforderten Listen schreibt der Compiler in eine Listendatei mit dem Namen *basisname*.lst.
basisname ist der Name der Quelldatei ohne die Dateiverzeichnisbestandteile und ohne das Suffix .cbl oder .cob. Die Listendatei wird in das aktuelle Dateiverzeichnis geschrieben.

Beispiel

```
-P "(ALL,NOXREF)"
```

13.3.4 Optionen für den Bindelauf

Die folgenden Optionen für den Binder bleiben ohne Wirkung, wenn durch Angabe der Option `-c` der Compilerlauf nach der Übersetzung beendet wird. Für jede solche ungenutzte Option gibt das `cobol`-Kommando eine Warnungsmeldung aus.
Hinweise zum Binden allgemein und zur Binde-Reihenfolge finden Sie im Abschnitt 13.1.2 (S.277ff).

–L *dateiverzeichnis*

Mit dieser Option können Pfadnamen von Dateiverzeichnissen angegeben werden, die der Binder nach Bibliotheken mit dem Namen `lib`*name*`.a` durchsuchen soll. Diese Bibliotheken müssen mit dem Operanden `-l` *name* angegeben werden.
Standardmäßig werden nur die Dateiverzeichnisse `/usr/lib` und `/usr/ccs/lib` nach den Bibliotheken durchsucht.
Die Reihenfolge der `-L`-Optionen ist signifikant. Die mit `-L` angegebenen Dateiverzeichnisse werden vorrangig vor den Standard-Dateiverzeichnissen durchsucht.

–M *name*

Mit *name* muß der PROGRAM-ID-Name des COBOL-Hauptprogramms in Großbuchstaben angegeben werden. Die Angabe dieser Option ist immer erforderlich, wenn das Hauptprogramm ein COBOL-Programm ist.

–o *ausgabedatei*

Die vom Binder erzeugte ausführbare Datei wird in die Datei *ausgabedatei* geschrieben. Enthält *ausgabedatei* keine Dateiverzeichnisbestandteile, wird die Datei in das aktuelle Dateiverzeichnis geschrieben, sonst in das mit *ausgabedatei* angegebene Dateiverzeichnis. Standardmäßig wird die ausführbare Datei unter dem Namen `a.out` in das aktuelle Dateiverzeichnis geschrieben.

–l *name*

Diese Option veranlaßt den Binder, beim Auflösen von Externverweisen per Autolink die Bibliothek mit dem Namen `lib`*name*`.a` zu durchsuchen.
Wenn mit der Binder-Option `-L` kein anderes Dateiverzeichnis angegeben wird, sucht der Binder die angegebene Bibliothek in den Standard-Dateiverzeichnissen `/usr/lib` und `/usr/ccs/lib`.
Die Sortierbibliothek lib*sort.a* (z.B.) ist nicht in den Standard-Dateiverzeichnissen, sondern als PLAM-Bibliothek im BS2000 installiert.
Die Bibliotheken werden vom Binder in der Reihenfolge durchsucht, in der sie in der Kommandozeile angegeben werden.

–l BLSLIB

Diese Option veranlaßt den Binder, PLAM-Bibliotheken zu durchsuchen, die mit den Shell-Umgebungsvariablen BLSLIB*nn* ($00 \leq nn \leq 99$) zugewiesen wurden. Die Umgebungsvariablen müssen vor Aufruf des Compilers mit den Bibliotheksnamen versorgt und mit dem POSIX-Kommando `export` exportiert werden. Die Bibliotheken werden in aufsteigender Reihenfolge *nn* durchsucht.
Alle mit den BLSLIB*nn*-Umgebungsvariablen zugewiesenen Bibliotheken werden intern in einer einzigen RESOLVE-Anweisung als Liste an den BINDER übergeben.

Beispiel

```
export BLSLIB00='$RZ99.SYSLNK.COB.999'
export BLSLIB01='$MYTEST.LIB'
cobol mytest.o -l BLSLIB -M MYTEST
```

13.3.5 Testhilfe-Option

–g

Der Compiler erzeugt zusätzliche Informationen (LSD) für die Testhilfe AID.
Standardmäßig werden keine Testhilfeinformationen erzeugt.
Diese Option wird intern auf COMOPT SYMTEST=ALL abgebildet. Die COMOPT SYMTEST sollte deshalb nicht mit -C übergeben werden.

13.3.6 C-spezifische Option

–CC *option*

option wird an den C-Compiler (c89-Kommando) weitergereicht.
Wenn die zu übergebende Option ein Argument enthält, darf (entgegen der üblichen Optioneneingabe) zwischen dem Optionsnamen und dem Argument kein Leerzeichen stehen.
Die Optionen des c89-Kommandos sind ausführlich im Handbuch „POSIX-Kommandos des C- und des C++-Compilers“ [33] beschrieben.

Beispiel

```
-CC -Xls -CC -FIabs
```

13.3.7 Eingabedateien

Der Compiler schließt aus der Endung des Dateinamens auf den Inhalt und führt die jeweils erforderlichen Übersetzungsschritte aus. Der Dateiname muß daher das Suffix enthalten, das gemäß den POSIX-Konventionen zum Datei-Inhalt paßt.
Folgende Konventionen gibt es:

suffix	**Bedeutung**
.cob/.cbl	COBOL-Quelldatei
.o	Objektdatei, erzeugt bei einer früheren Übersetzung
.a	Bibliothek mit Objektdateien, erzeugt mit dem Dienstprogramm `ar`
.c/.C/.i	C-Quelldatei (siehe Handbuch „POSIX-Kommandos des C- und des C++-Compilers“)

Die Dateien mit dem Suffix .cob oder .cbl sind die Eingabequellen für den COBOL85-Compiler. Der COBOL85-Compiler erkennt auch COBOL-Quelldateien, deren Namen nicht mit einem dieser Standard-Suffixe enden. Hierzu sind die Namen der Quelldateien nicht als Operanden, sondern mit der Option `-k` *dateiname* anzugeben (siehe S.283).

Die Dateien mit dem Suffix .o und .a sind die Eingabequellen für den Binder.

Die Dateinamen mit anderen Suffixen werden an das c89-Kommando für den C-Compiler weitergereicht.

13.3.8 Ausgabedateien

Folgende Dateien werden mit Standardnamen erzeugt und im aktuellen Dateiverzeichnis abgelegt. Für die Ausgabe des Binders (a.out) können mit der Option `-o` (siehe S.287) ein anderer Dateiname und ein anderes Dateiverzeichnis gewählt werden.
basisname ist der Name der Quelldatei ohne das Standard-Suffix und die Dateiverzeichnisbestandteile.

basisname.lst — Datei, die alle Übersetzungslisten enthält

basisname.o — vom Compiler erzeugte LLM-Objektdatei, die mit dem Binder weiterverarbeitet werden kann

a.out — vom Binder erzeugte ausführbare Datei

Bei der Übersetzung von Quellprogramm-Folgen werden die Namen der LLM-Objektdateien für das zweite bis letzte Quellprogramm aus dem PROGRAM-ID-Namen und dem Suffix .o gebildet.

13.4 Einführungsbeispiele

Übersetzen und Binden mit dem cobol-Kommando

```
cobol -M BSPPROG hugo.cob
```

übersetzt `hugo.cob` und erzeugt eine ausführbare Datei `a.out`.
Das Programm mit dem PROGRAM-ID-Namen BSPPROG wird zum Hauptprogramm.

```
cobol -o hugo -M BSPPROG hugo.cob
```

übersetzt `hugo.cob` und erzeugt eine ausführbare Datei `hugo`.
Das Programm mit dem PROGRAM-ID-Namen BSPPROG wird zum Hauptprogramm.

```
cobol -c -P "(SOURCE,DIAG)" hugo.cob upro.cob
```

übersetzt `hugo.cob` und `upro.cob`, erzeugt die Objektdateien `hugo.o` und `upro.o` sowie für beide Programme je eine Quellprogramm- und eine Fehlerliste.
Die Listen werden in den Listendateien `hugo.lst` bzw. `upro.lst` abgelegt.

```
cobol -M BSPPROG -o hugo hugo.o upro.o
```

bindet das Hauptprogramm `hugo.o` und das Unterprogramm `upro.o` zu einer ausführbaren Datei `hugo`.
Das Programm mit dem PROGRAM-ID-Namen BSPPROG wird zum Hauptprogramm.

13.5 Unterschiede zu COBOL85 im BS2000

Wegen der systemspezifischen Unterschiede zwischen POSIX und BS2000 sind bei der Entwicklung von COBOL-Programmen, die in POSIX ablaufen sollen, einige Besonderheiten hinsichtlich Sprachumfang und Ablaufverhalten zu beachten, die im folgenden aufgeführt sind.

13.5.1 Sprachfunktionale Einschränkungen

Die im folgenden aufgeführten Sprachmittel des COBOL85-Compilers werden bei Programmablauf im POSIX-Subsystem nicht unterstützt:

Dynamischer Unterprogrammaufruf

Der Aufruf eines Unterprogramms mit der COBOL-Anweisung CALL *bezeichner* ist in POSIX nicht möglich.
Das Laufzeitsystem quittiert einen dynamischen Unterprogrammaufruf mit der Fehlermeldung COB9164 (CALL nicht ausführbar).

ENTRY-Anweisung

Die ENTRY-Anweisung ist bei Programmablauf unter POSIX nicht zulässig, da mit ihr nur Einsprungstellen auf Objektmodule definiert werden können, der Compiler unter POSIX aber grundsätzlich Bindelademodule (LLMs) erzeugt.

Segmentierung

Da der Compiler unter POSIX grundsätzlich Bindelademodule (LLMs) erzeugt, ist die Segmentierung von COBOL-Programmen in POSIX nicht möglich.

Dateiverarbeitung

- Die Kennsatzbehandlung bei der Verarbeitung von Magnetbändern ist in POSIX nicht möglich.
- Fixpunktausgabe für Wiederanlauf von Magnetbändern ist in POSIX nicht möglich.
- Simultanverarbeitung von Dateien ist in POSIX nicht möglich.

13.5.2 Sprachfunktionale Erweiterungen

Zugriff auf Kommandozeile

Bei Ablauf in POSIX kann vom Programm aus mittels ACCEPT-/DISPLAY-Anweisungen in Verbindung mit den Sondernamen ARGUMENT-NUMBER und ARGUMENT-VALUE auf die Kommandozeile zugegriffen werden (siehe COBOL85-Sprachbeschreibung [1]).

Beispiel

```
IDENTIFICATION DIVISION.
...
SPECIAL-NAMES.
    ARGUMENT-NUMBER IS NO-OF-CMD-ARGUMENTS
    ARGUMENT-VALUE  IS CMD-ARGUMENT
...
WORKING-STORAGE SECTION.
01  I    PIC  99   VALUE 0.
01  J    PIC  99   VALUE 0.
01  A    PIC  X(5) VALUE ALL "x".
...
PROCEDURE DIVISION.
...
  ACCEPT I FROM NO-OF-CMD-ARGUMENTS
  DISPLAY "no. of command arguments=" I
  PERFORM VARYING J FROM 1 BY 1 UNTIL J > I
    ACCEPT A FROM CMD-ARGUMENT
    DISPLAY "cmd argument-" J " <" A ">"
  END-PERFORM
...
 DISPLAY 2 UPON NO-OF-CMD-ARGUMENTS
 ACCEPT  A  FROM CMD-ARGUMENT
 DISPLAY "argument-2" " :" A ":"
...
```

Programmaufruf

```
a.out AAAA BBB CC D
```

Ablaufprotokoll

```
no. of command arguments=4
cmd argument-1 <AAAA >
cmd argument-2 <BBB  >
cmd argument-3 <CC   >
cmd argument-4 <D    >
argument-2 :BBB  :
```

13.5.3 Unterschiede bezüglich der Programm-Betriebssystem-Schnittstellen

Für COBOL-Programme, die in POSIX ablaufen, ist in einigen Bereichen ein gegenüber dem Ablauf im BS2000 abweichendes Verhalten zu beachten:

Ein-/Ausgabe geringer Datenmengen

Den COBOL85-Herstellernamen in ACCEPT-/DISPLAY-Anweisungen zur Ein-/Ausgabe kleiner Datenmengen sind in POSIX folgende Standard-Ein-/Ausgabeströme zugeordnet:

COBOL85	BS2000	POSIX
TERMINAL	SYSDTA	stdin
SYSIPT	SYSIPT	undefiniert
TERMINAL	SYSOUT	stdout
PRINTER	SYSLST	stdout
PRINTER01..99	SYSLST01..99	undefiniert
SYSOPT	SYSOPT	undefiniert
CONSOLE	CONSOLE	undefiniert

Sortieren und Mischen

Die Sortierdatei wird automatisch im BS2000-Dateisystem abgelegt, und der POSIX-Nutzer hat auf sie keinen Zugriff.

Jobvariablen

Die Verwendung von BS2000-Jobvariablen ist bei Programmablauf in POSIX nicht möglich.

Auftrags- und Benutzerschalter

Die Verwendung von BS2000-Auftrags- und Benutzerschaltern ist bei Programmablauf in POSIX nicht sinnvoll.

Dateiverarbeitung

– Die Verknüpfung zwischen dem externen Dateinamen in der ASSIGN-Klausel und dem Dateinamen im POSIX-Dateisystem wird über eine Umgebungsvariable hergestellt, deren Name identisch mit dem externen Dateinamen in der ASSIGN-Klausel ist.
Der Name der Umgebungsvariablen muß immer in Großbuchstaben geschrieben werden. Ausführliche Informationen hierzu finden Sie in Abschnitt 13.6.2, S.298ff.

– Nach einem erfolglosen OPEN INPUT auf eine Datei, für die nicht OPTIONAL angegeben wurde, wird der Programmablauf nicht unterbrochen.

– Einige Werte des Ein-/Ausgabezustands verändern sich in POSIX:

BS2000	**POSIX**
37	30
93, 94, 95	90

– Im erweiterten Ein-/Ausgabezustand, der sich in der FILE STATUS-Klausel mit dateiname-2 anfordern läßt, wird statt des (BS2000-) DVS-Codes der (POSIX-) SIS-Code ausgegeben.

– Die Dateiattribute werden beim ersten Öffnen der Datei endgültig festgelegt und können später nicht mehr geändert werden.

– Relative Dateien, die die BS2000-Zugriffsmethode UPAM verwenden, können nicht verarbeitet werden.

13.6 Verarbeiten von POSIX-Dateien

13.6.1 Programmablauf in BS2000-Umgebung

Ein COBOL-Programm, das im BS2000 entwickelt und zum Ablauf gebracht wird, kann unter bestimmten Voraussetzungen außer katalogisierten (BS2000-)Dateien auch Dateien aus dem POSIX-Dateisystem verarbeiten.

Voraussetzungen

– Beim Übersetzen muß die Compileroption ENABLE-UFS-ACCESS=YES bzw. die SDF-Option RUNTIME-OPTIONS=PAR(ENABLE-UFS-ACCESS=YES) angegeben werden.

– Beim Binden muß das in der CRTE-Bibliothek SYSLNK.CRTE.POSIX enthaltene POSIX-Bindeschalter-Modul eingebunden werden, und zwar **vorrangig** vor den Modulen in der Bibliothek SYSLNK.CRTE bzw. SYSLNK.CRTE.PARTIAL-BIND.
Beim Binden mit TSOSLNK oder BINDER sollte diese Bibliothek mit einer INCLUDE- bzw. INCLUDE-MODULES-Anweisung (ohne Angabe des Modulnamens) eingebunden werden.
Beim dynamischen Binden mit dem DBL muß der Bibliothek eine BLSLIB*nn* mit niedrigerer *nn* zugewiesen werden als den nachrangig einzubindenden CRTE-Bibliotheken.
Bei Programmentwicklung in der POSIX-Shell mit dem cobol-Kommando wird die CRTE-Bibliothek automatisch eingebunden.

Einschränkungen

Die Verarbeitung einer BS2000- oder POSIX-Datei unterliegt folgenden Einschränkungen:

– keine Kennsatzbehandlung möglich

– keine Fixpunktausgabe für Wiederanlauf möglich

– keine Simultanverarbeitung möglich

– Die Dateiattribute werden beim ersten Öffnen der Datei endgültig festgelegt und können später nicht mehr geändert werden.

– Relative Dateien, die die BS2000-Zugriffsmethode UPAM verwenden, können nicht verarbeitet werden.

Der erweiterte Ein-/Ausgabezustand enthält statt des (BS2000-) DVS-Code den (POSIX-) SIS-Code (siehe S.299).

Zuweisen einer POSIX-Datei

Die Zuweisung einer POSIX-Datei erfolgt mit einer SDF-P-Variablen namens SYSIOL-externer-name, wobei SYSIOL- ein fester Namensbestandteil ist und externer-name den Linknamen aus der ASSIGN-Klausel des Programms enthalten muß. externer-name darf keine Kleinbuchstaben enthalten.

Die SDF-P-Variable wird mit dem Kommando SET-VARIABLE folgendermaßen initialisiert:

```
/[SET-VAR] SYSIOL-externer-name= { '*POSIX(dateiname)'           }
                                 { '*POSIX(relativer-pfadname)' }
                                 { '*POSIX(absoluter-pfadname)' }
```

`dateiname` bezeichnet die angeforderte POSIX-Datei, wenn sie im Home-Verzeichnis des POSIX-Dateisystems steht.

`relativer-pfadname` ist der Dateiname mit den Dateiverzeichnisbestandteilen ab dem Home-Verzeichnis.

`absoluter-pfadname` ist der Dateiname mit allen Dateiverzeichnisbestandteilen einschließlich Root-Verzeichnis (Beginn mit /).

Beispiel für gemischte Dateiverarbeitung

COBOL-Quellprogramm:

```
...
FILE-CONTROL.
    SELECT POSFILE ASSIGN TO "CUST1"
    SELECT BS2FILE ASSIGN TO "CUST2"
...
```

Zuweisung der POSIX-Datei vor Aufruf des Programms:

```
/[SET-VAR] SYSIOL-CUST1='*POSIX(/USERIDXY/customers/cust1)'
```

Zuweisung der BS2000-Datei vor Aufruf des Programms:

```
/SET-FILE-LINK CUST2,CUST.FILE
```

13.6.2 Programmablauf in der POSIX-Shell

Ein COBOL-Programm, das in der POSIX-Shell oder im BS2000 entwickelt und zum Ablauf gebracht wird, kann POSIX-Dateien ohne besondere Maßnahmen beim Übersetzen und Binden (vgl. Programmablauf im BS2000) verarbeiten.
Die Verarbeitung von BS2000-Dateien aus der POSIX-Shell ist nicht möglich.

Bei der Verarbeitung von POSIX-Dateien gelten sprachfunktionale Einschränkungen gegenüber der Dateiverarbeitung im BS2000 (siehe S.292).

Zuweisen einer POSIX-Datei

Die Zuweisung einer POSIX-Datei erfolgt mit einer Shell-Umgebungsvariablen namens `externer-name`.
`externer-name` ist der Dateiname aus der ASSIGN-Klausel im Programm. Er darf keine Kleinbuchstaben enthalten.
Die Umgebungsvariable muß mit dem Namen der POSIX-Datei initialisert und mit dem POSIX-Kommando export exportiert werden.
Die Umgebungsvariable wird folgendermaßen initialisiert:

```
externer-name= { dateiname           }
               { relativer-pfadname  }
               { absoluter-pfadname  }
```

`dateiname` bezeichnet die angeforderte POSIX-Datei, wenn sie im aktuellen Dateiverzeichnis steht. Der Dateiname darf nicht mit einem Bindestrich beginnen.

`relativer-pfadname` ist der Dateiname mit den Dateiverzeichnisbestandteilen ab dem aktuellen Verzeichnis.

`absoluter-pfadname` ist der Dateiname mit allen Dateiverzeichnisbestandteilen einschließlich Root-Verzeichnis (Beginn mit /).

Beispiel

COBOL-Quellprogramm:

```
...
FILE-CONTROL.
SELECT AFILE ASSIGN TO "CUST1"
...
```

Verknüpfung mit der POSIX-Datei cust1 vor Aufruf des Programms:

```
export CUST1=/USERIDXY/customers/cust1
```

13.6.3 Ein-/Ausgabezustände

Jeder Datei im Programm können mit der FILE STATUS-Klausel Datenfelder zugeordnet werden, in denen das Laufzeitsystem nach jedem Zugriff auf die Datei Informationen darüber hinterlegt,

– ob die Ein-/Ausgabeoperation erfolgreich war und

– welcher Art ggf. die dabei aufgetretenen Fehler sind.

Diese Informationen können z.B. in den DECLARATIVES durch USE-Prozeduren ausgewertet werden und gestatten eine Analyse von Ein-/Ausgabefehlern durch das Programm. Als Erweiterung zum COBOL-Standard bietet COBOL85 die Möglichkeit, in diese Analyse auch die Schlüssel der POSIX-Fehlermeldungen einzubeziehen. Dadurch läßt sich eine feinere Differenzierung der Fehlerursachen erreichen. Die FILE STATUS-Klausel wird im FILE-CONTROL-Paragraphen der ENVIRONMENT DIVISION angegeben; ihr Format ist z.B. in Abschnitt 9.2.9 (S.194f) dargestellt.
Die beiden in der FILE STATUS-Klausel definierbaren Datenfelder haben folgende Funktion:

datenname-1

enthält nach jeder Ein-/Ausgabeoperation auf die zugeordnete Datei einen zweistelligen numerischen Zustandscode.

datenname-2

ist unterteilt in datenname-2-1 und datenname-2-2. Es dient der Aufnahme des (POSIX-) SIS-Codes zum jeweiligen Ein-/Ausgabezustand und enthält nach jedem Zugriff auf die zugeordnete Datei einen Wert, der vom Inhalt des Feldes datenname-1 abhängt und sich aus folgender Zusammenstellung ergibt:

Inhalt von datenname-1 ungleich 0?	SIS-Code ungleich 0?	Wert von datenname-2-1	Wert von datenname-2-2
nein	nicht relevant	undefiniert	undefiniert
ja	nein	0	undefiniert
ja	ja	96	SIS-Code der zugeordneten Fehlermeldung

Bei Programmablauf im BS2000 läßt sich der Bedeutungstext des jeweiligen SIS-Codes mit dem Kommando HELP-MSG-INFORMATION SIS<datenname-2-2> ausgeben.

Der einfache und der erweiterte Ein-/Ausgabezustand sind in den beiden folgenden Tabellen beschrieben.

Einfacher Ein-/Ausgabezustand

Wert	Org*)	Bedeutung
0x		erfolgreiche Ausführung
00	SRI	keine weitere Information
02	I	erfolgreicher READ, erlaubter doppelter Schlüssel
04	SRI	erfolgreicher READ, aber Satzlängenfehler
05	SRI	erfolgreicher OPEN auf nicht vorhandene OPTIONAL-Datei
07	S	- erfolgreicher OPEN mit NO REWIND - erfolgreicher CLOSE mit NO REWIND, REEL/UNIT oder FOR REMOVAL
1x		erfolglose Ausführung: AT END-Bedingung
10	SRI	erfolgloser READ, da Dateiende erreicht
14	R	erfolgloser READ, da Schlüsselfeldlängenfehler
2x		erfolglose Ausführung, Schlüsselfehler
21	I	falsche Schlüsselreihenfolge bei sequentiellem Zugriff
22	RI	WRITE auf schon vorhandenen Satz
23	RI	READ auf nicht vorhandenen Satz
24	RI	Schlüsselfeldlängenfehler
3x		erfolglose Ausführung, permanenter Fehler
30	SRI	keine weitere Information (SIS-Code beachten)
34	S	unzureichende Sekundärzuweisung im SET-FILE-LINK-Kommando
35	SRI	OPEN INPUT/I-O auf nicht vorhandene Datei
38	SRI	OPEN auf eine mit CLOSE WITH LOCK geschlosseneDatei
39	SRI	OPEN-Fehler wegen falscher Dateimerkmale
4x		erfolglose Ausführung, logischer Fehler
41	SRI	OPEN auf bereits geöffnete Datei
42	SRI	CLOSE auf nicht geöffnete Datei
43	S	REWRITE ohne vorherigen erfolgreichen READ
	RI	DELETE/REWRITE ohne vorherigen erfolgreichen READ
44	SRI	WRITE/REWRITE mit unzulässiger Satzlänge
46	S	erneuter READ nach erfolglosem READ oder erkanntem AT END
	RI	sequent. READ nach erfolglosem READ/START oder nach erkanntem AT END
	S	READ auf nicht zum Lesen geöffnete Datei
47	RI	READ/START auf nicht zum Lesen geöffnete Datei
	SRI	WRITE auf nicht zum Schreiben geöffnete Datei
48	S	REWRITE auf nicht mit I-O geöffnete Datei
49	RI	DELETE/REWRITE auf nicht mit I-O geöffnete Datei
9x		sonstige erfolglose Ausführung
90	SRI	Systemfehler, keine weiteren Informationen
91	SRI	OPEN-Fehler oder kein freies Gerät

*) S = sequentielle Organisation, R = relative Organisation, I = indexsequentielle Organisation

Erweiterter Ein-/Ausgabezustand (SIS-Code)

Ein-/Ausgabezustand	Bedeutung
0601	Dateiende ist erreicht
0602	Spezifizierter Satz existiert nicht
0603	Spezifizierter Satz existiert bereits
0604	Dateianfang ist erreicht
0605	Spezifizierter Link existiert nicht
0606	Dateiname ist länger als P_MAXFILENAME
0607	Pfad ist länger als P_MAXPATHSTRG
0608	Pfadname ist länger als P_MAXPATHNAME
0609	Linkname ist länger als P_MAXLINKNAME
0610	kein ausreichender Speicherplatz verfügbar
0611	Anzahl der Pfadelemente übersteigt P_MAXHIERARCHY
0612	Funktion wird nicht unterstützt
0613	Dateiname ist fehlerhaft oder leer
0614	Anzahl der Sekundärschlüssel übersteigt P_MAXKEYS
0615	Anzahl offener Dateien übersteigt systemspezifische Grenze
0616	Spezifizierte Datei existiert nicht
0617	Kein Schreibzugriff erlaubt
0618	kein Dateiname spezifiziert
0619	Datei ist gesperrt
0620	unzulässige Kombination von Dateiattributen
0621	File-Handle ist ungültig
0622	Aktueller Datensatz ist kürzer als MINSIZE
0623	Aktueller Datensatz ist länger als MAXSIZE
0625	Vor rwrite sequentiell wurde kein read sequentiell ausgeführt
0626	Spezifiziertes Satzformat ist unzulässig
0627	MINSIZE ist größer als MAXSIZE
0628	Spezifizierte Organisation ist unzulässig
0629	Nicht existent spezifizierte Datei existiert
0630	Spezifizierte Zugriffsfunktion ist nicht erlaubt
0631	Spezifizierter Schlüssel ist unzulässig
0632	Mehrfachschlüssel ist nicht erlaubt
0633	Aktueller Satz ist zur Zeit gesperrt

Ein-/Ausgabezustand	Bedeutung
0634	Aktueller Schlüssel in fehlerhafter Reihenfolge
0635	Spezifizierter Pfad ist undefiniert
0636	Es ist ein systemspezifischer Fehler aufgetreten
0637	Zeilenende ist erreicht
0638	Satz wurde abgeschnitten
0640	Kein Speicherplatz zur Dateierweiterung verfügbar
0643	Spezifizierter Öffnungsmodus ist unzulässig
0644	Länge des Links übersteigt P_MAXLINKSTRG
0645	Versionsidentifikation ist fehlerhaft
0646	Spezifizierte Dateiexistenz ist unzulässig
0647	Syntaxfehler im Dateinamen, Link oder Pfad
0649	Spezifizierter Modus beim Schließen ist unzulässig
0650	Dateizugriff ist nicht erlaubt
0651	Fehlerhafter Parameter angegeben
0652	Zeiger in den Ein-/Ausgabebereich ist fehlerhaft
0653	Satzlänge ist fehlerhaft
0654	Speichermangel auf Ausgabemedium aufgetreten
0655	Spezifizierte Vorschubsteuerung ist unzulässig
0656	Spezifizierter Code ist unzulässig
0657	Öffnungsmodus und Dateiexistenz sind unzulässig kombiniert
0658	Ein-/Ausgabeunterbrechung aufgetreten
0659	Länge des Schlüsselwortes übersteigt P_MAXKEYWORD
0660	Schlüsselwort ist mehrdeutig
0661	Anzahl der Exits übersteigt P_MAXEXITS
0662	Zeilenvorschubsteuerzeichen erkannt
0663	Seitenvorschubsteuerzeichen erkannt
0664	Einige Pfade sind nicht geschlossen
0665	Nächster Satz hat den gleichen Sekundärschlüssel
0666	Sekundärschlüssel des geschriebenen Satzes existiert bereits
0667	Aktuelle Satznummer ist größer als MAX_REC_NR
0668	Pfad ist bereits definiert
0669	Link ist bereits definiert
0670	Spezifizierter Wert für Positionierbedingung ist unzulässig

Ein-/Ausgabezustand	Bedeutung
0671	Unbekanntes Kontrollzeichen gefunden
0672	Es konnte kein eindeutiger Dateiname erzeugt werden
0673	Letzter Teilsatz wurde nicht abgeschlossen
0674	Spezifizierter Wert für Positionierung ist unzulässig
0675	Satzformat ist nicht bestimmbar
0676	MAXSIZE ist nicht bestimmbar
0677	Interner PROSOS-D Fehler aufgetreten
0678	Spezifizierte Datei ist ein Container von Dateien
0679	Spezifizierte Datei ist unter angegebenem Pfad nicht erreichbar
0680	Versionsangabe kann nicht erhöht werden
0681	Nochmaliges Öffnen nach implizitem Schließen wurde abgewiesen
0682	Fehler bei Initialisierung von PROSOS-D
0683	Linkindirektionen übersteigen P_MAXLINKNESTING

14 Nutzbare Software für COBOL-Anwender

14.1 Advanced Interactive Debugger AID

Charakterisierung des Produktes

AID ist ein leistungsstarkes Testsystem zur Fehler-Diagnose, zum Test und für die vorläufige Korrektur von Programm-Fehlern im BS2000.

AID unterstützt das symbolische Testen von COBOL-, C-, C++, Assembler-, FORTRAN- und PL/1-Programmen und das nicht-symbolische Testen auf Maschinencode-Ebene aller Programmiersprachen des BS2000.

Beim symbolischen Testen eines COBOL-Programms können die symbolischen Namen aus einem COBOL-Quellprogramm zur Adressierung verwendet werden. Das nicht-symbolische Testen auf Maschinencode-Ebene bietet sich dort an, wo das symbolische Testen nicht ausreicht oder wegen fehlender Testhilfe-Information nicht möglich ist.

Über spezielle AID-Kommandos sind u.a. folgende Grundfunktionen aufrufbar:

- zur Ablaufüberwachung
  - bestimmter Quellprogramm-Anweisungstypen
  - ausgewählter Ereignisse im Programmablauf
  - vereinbarter Programmadressen
- zum Zugriff auf Datenfelder und Modifikation von Feldinhalten
- zur Verwaltung von AID-Ausgabedateien und Bibliotheken
- zur Festlegung globaler Vereinbarungen
- zur Steuerung von
  - Ausgabe-Datenmengen
  - AID-Ausgabemedien

Die Hantierung wird unterstützt durch eine zusätzliche HELP-Funktion

- für alle AID-Kommandos und Operanden
- für die Bedeutung und die möglichen Reaktionen auf AID-Meldungen.

Der Anwender kann festlegen, daß AID den Programmablauf an definierten Adressen oder bei Ausführung ausgewählter Anweisungstypen oder beim Eintreten definierter Ereignisse unterbricht und dann Subkommandos ausführt. Ein Subkommando ist ein einzelnes Kommando oder eine Folge von AID- und BS2000-Kommandos. Es wird als Operand eines AID-Kommandos definiert. Ab der Version V2.0 kann die Ausführung von Subkommandos von Bedingungen abhängig gemacht werden. Damit lassen sich u.a. Programmzustände bzw. Variablenwerte dynamisch überwachen.

Außerdem können Datenfelder modifiziert und Datenelemente, Datengruppen oder ganze DATA DIVISIONS von COBOL-Programmen ausgegeben werden.

Mit einem Kommando kann man sich anzeigen lassen, auf welcher Stufe der Aufrufhierarchie das Programm unterbrochen wurde und welche Module in der CALL-Verschachtelung liegen.

Mit AID kann ein laufendes Programm bearbeitet oder ein Speicherauszug in einer Plattendatei diagnostiziert werden. Innerhalb einer Testsitzung kann zwischen beiden Möglichkeiten gewechselt werden, z.B. um Datenbestände im laufenden Programm mit einem Speicherauszug zu vergleichen.

Beschreibung der Funktionen

AID dient zum Test und zur Diagnose von Anwenderprogrammen auf Primärsprachebene (High Level Language Testhilfe).

Die Funktionen für Test und Diagnose auf Primärsprachebene von COBOL-Programmen, die mit dem COBOL85 übersetzt wurden, sind:

– Ausgeben und Setzen von benutzerdefinierten Daten

  Daten, die im Benutzerprogramm definiert sind, können interaktiv angesprochen werden. Dabei gelten die Regeln für Qualifizierung, Eindeutigkeit, Indizierung und Bereichsgrenzen von COBOL.
  Die Daten selbst werden entsprechend den im Benutzerprogramm angegebenen Attributen konvertiert und aufbereitet.

– Symbolischer Dump

  Alle oder ausgewählte Daten von Programmen der dynamischen Programmverschachtelung können entsprechend dieser Programmverschachtelung aufbereitet ausgegeben werden.

– Setzen von Testpunkten

  Über die Sourcereferenz oder die Marken im Programm (Paragraphen, Kapitel) können Testpunkte, an denen bestimmte Aktionen ausgeführt werden, gesetzt und rückgesetzt werden. Das Ansprechen der Marken erfolgt nach den in COBOL geltenden Qualifizierungsregeln.

– Ablaufverfolgung auf Statementebene

  Dynamische Ablaufverfolgung steuerbar über Statementklassifikation (z.B. Procedure trace, Controlflow trace, Assignment trace...) wird unterstützt. Ausgegeben werden bei AID die Sourcereferenz der durchlaufenen Statements, die der Statementklassifikation entsprechen.

Dokumentation

AID Basis-Handbuch [22]

AID Testen von COBOL-Programmen [8]

AID Testen auf Maschinencodeebene [23].

14.2 Library Maintenance System LMS

Charakterisierung des Produktes

Das Bibliotheksverwaltungssystem LMS erstellt und verwaltet Programmbibliotheken und bearbeitet die darin enthaltenen Elemente.

Programmbibliotheken sind PAM-Dateien des BS2000, die mit der Bibliotheks-Zugriffsmethode PLAM (Program Library Access Method) bearbeitet werden. Daher werden sie auch als PLAM-Bibliotheken bezeichnet.

Der grundlegende Nutzen besteht darin, daß

- alle Elementtypen in einer Bibliothek mit einheitlichen Anweisungen bearbeitet werden können,
- gleichnamige Elemente existieren können, die sich nur durch Typ- oder Versionsbezeichnung unterscheiden,
- Versionsbezeichnungen automatisch erhöht werden,
- auf die Bibliothek von mehreren Benutzern simultan auch schreibend zugegriffen werden kann,
- differenzierte Zugriffsrechte je Element vergeben werden können,
- der Zugriff auf Elemente überwacht werden kann,
- für die meisten während eines SW-Entwicklungsprozesses anfallenden Datenelemente eine einheitliche Datenhaltung mit einheitlichen Zugriffsfunktionen existiert und
- die Dienstprogramme und Compiler auf diese Datenhaltung zugreifen und die einzelnen Elemente direkt verarbeiten können.

Damit unterstützt LMS die Programmerstellung, -pflege und -dokumentation.

Struktur der Bibliotheken

Eine Programmbibliothek ist eine Datei mit Unterstruktur. Sie enthält Elemente und ein Inhaltsverzeichnis der gespeicherten Elemente.

Ein Element ist eine logisch zusammengehörige Datenmenge, z.B. eine Datei, eine Prozedur, ein Bindemodul oder ein Quellprogramm. Jedes Element in der Bibliothek ist einzeln ansprechbar.

Jede Bibliothek hat einen Eintrag im Systemkatalog. Der Benutzer kann den Namen und andere Dateimerkmale, wie z.B. die Schutzfrist oder die gemeinsame Benutzbarkeit festlegen.

Das Speichern mehrerer Dateien in einer Bibliothek entlastet den Systemkatalog, da dort nur die Bibliothek eingetragen ist und nicht jedes Element. Außerdem spart es Speicherplatz, da die Elemente in der Bibliothek in komprimierter Form abgespeichert werden.

Unterstützung mehrerer Versionen

Bei Verwendung der Delta-Technik werden von mehreren Versionen eines Elements nur die Unterschiede (Deltas) zur Vorgängerversion abgespeichert, was zusätzlich Speicherplatz sparen hilft.

Beim Lesen solcher Deltaversionen werden diese Deltas von LMS wieder an die entsprechenden Stellen eingemischt. Dem Anwender steht somit wieder das komplette Element zur Verfügung.

LMS unterstützt symbolische Versionsbezeichner und erhöht Versionen automatisch in Abhängigkeit vom gewählten Versionsformat.

Einbettung in die Programmierumgebung

Die Dienstprogramme der Programmierumgebung, wie EDT, Compiler etc., können direkt auf Programmbibliotheken zugreifen.

Dokumentation

LMS Beschreibung [10]

14.3 Jobvariablen

Charakterisierung des Produktes

Jobvariablen sind Datenobjekte zum Austausch von Informationen zwischen Benutzern einerseits und Betriebssystem und Benutzern andererseits.

Der Benutzer kann Jobvariablen einrichten und verändern. Er kann das Betriebssystem anweisen, beim Eintreten gewisser Ereignisse bestimmte Jobvariablen auf vereinbarte Werte zu setzen.

Jobvariablen sind ein flexibles Werkzeug zur Auftragssteuerung unter Benutzerkontrolle. Sie bieten die Möglichkeit, Abhängigkeiten von komplexen Produktionsabläufen einfach zu definieren und bilden die Basis für eine ereignisgesteuerte Auftragsverarbeitung.

Beschreibung der Funktionen

Jobvariablen sind vom Betriebssystem verwaltete Objekte, die über Namen adressiert werden und in die Daten bis zu einer Länge von 256 byte abgespeichert werden können. Sie dienen zum Austausch von Informationen zwischen Benutzern einerseits sowie Betriebssystem und Benutzern andererseits. Auf sie kann über die Kommando- und Makroschnittstelle zugeriffen werden. Bei Verwendung der Komponente SDF der BS2000-BC können Jobvariablen als globale Parameter auf Kommandoebene verwendet werden.

In Bedingungsanweisungen kann man Jobvariablen über boolesche Operationen verknüpfen und somit die Ausführung einzelner Aktionen vom Wahrheitswert der Bedingung abhängig machen. Benutzer-Jobvariablen und überwachende Jobvariablen (s.u.) bieten zudem die Möglichkeit der synchronen und asynchronen Ereignissteuerung auf Kommando- und Programmebene.

Für die verschiedenen Aufgabengebiete gibt es unterschiedliche Jobvariablen:

- Benutzer-Jobvariablen

Die allgemeinste Form, in der Jobvariablen angeboten werden, ist die Form der Benutzer-Jobvariablen. Ihr Name, ihre Lebensdauer und die abzuspeichernden Daten werden ausschließlich vom Benutzer bestimmt. Sie kann mit Schutzattributen wie Paßwörtern, Schreibschutz und Verfallsdatum versehen werden. Der Zugriff auf sie kann auf eine Benutzerkennung beschränkt oder generell gestattet sein.

Benutzer-Jobvariablen sind besonders geeignet zum Austausch von Informationen. Sie können aber auch zur Auftragssteuerung verwendet werden.

- Überwachende Jobvariablen

Die überwachende Jobvariable ist eine Spezialform der Benutzer-Jobvariablen. Sie wird einem Auftrag oder Programm zugeordnet. Name, Lebensdauer und Schutzattribute bestimmt der Benutzer. Im Gegensatz zur Benutzer-Jobvariable wird sie aber vom Betriebssystem mit fest vorgegebenen Werten versorgt, die den Status des zugeordneten Auftrags oder Programms widerspiegeln.

Überwachende Jobvariablen sind besonders geeignet zur Auftragssteuerung, wie sie u.a. bei Abhängigkeiten in Produktionsabläufen notwendig ist.

Dokumentation

Jobvariablen Beschreibung [7]

14.4 Datenbankschnittstelle ESQL-COBOL

Charakterisierung des Produktes

ESQL-COBOL (BS2000/OSD) V2.0 realisiert für COBOL-Anwendungen im BS2000/OSD die Anwender-Programmschnittstelle "embedded SQL" zum Datenbanksystem SESAM/SQL-Server V2.0.
ESQL-COBOL V2.0 ist die Nachfolgeversion von ESQL-COBOL V1.1 für SESAM/SQL.

Für das Datenbanksystem UDS/SQL ist weiterhin ESQL-COBOL V1.1 zu verwenden!

Die neuartige Architektur des Übersetzungs- und Datenbanksystems ESQL-COBOL (BS2000/OSD) V2.0 / SESAM/SQL-Server V2.0 führt zu einer neuen Aufgabenverteilung zwischen Precompiler und Datenbanksystem.

Der erweiterte SQL-Funktionsumfang von SESAM/SQL-Server V2.0 kann mit ESQL-COBOL (BS2000/OSD) V2.0 uneingeschränkt genutzt werden:
Erweiterungen bezüglich Datenmanipulation, Datendefinition, Datenkontrolle, Utility- und Auskunftsfunktionen (Schema Information Tables).

Ferner bietet ESQL-COBOL (BS2000/OSD) V2.0
- abgestufte Syntax- und Semantikprüfungen der „embedded SQL“ zum Vorübersetzungszeitpunkt, wahlweise mit oder ohne Verbindung zur Datenbank.
- zur Fehlerbehandlung den standardisierten Returncode SQLSTATE des ISO-Standards von 1992, zusätzlich zu dem bisherigen SQLCODE des ISO-Standards von 1989. Dies verbessert deutlich die Portierbarkeit der SQL-Anwendungen.

ESQL-COBOL (BS2000/OSD) V2.0 ist lediglich als SQL-Precompiler zur Programmentwicklung erforderlich. Das SQL-Laufzeitsystem ist Bestandteil von SESAM/SQL-Server.

Für den Einsatz von ESQL-COBOL (BS2000/OSD) V2.0 - ob mit oder ohne Verbindung zur Datenbank - ist das SQL-Laufzeitsystem von SESAM/SQL V2.0 erforderlich.

Umfang der SQL-Funktionen

- Suchen von Datensätzen (SELECT-Anweisung) einschließlich höherer Funktionen wie Join, Arithmetik, Aggregatsfunktionen (z.B. Durchschnittsbildung),
- Neuaufnahme, Ändern, Löschen von Datensätzen.

ESQL-COBOL (BS2000) bietet auch einige über die ISO-Norm hinausgehende Funktionen an, um die bestehende Funktionalität der Datenbanksysteme abzubilden,
z.B. multiple Felder für SESAM/SQL, strukturierte Felder (Vektoren, Wiederholungs- und Datengruppen) für UDS/SQL.

Technische Hinweise

Die SQL-Anweisungen eines ESQL-COBOL-Programms sind in den COBOL-Code eingebettet und werden von einem Precompiler durch COBOL-CALL-Aufrufe ersetzt. Die Ausgabe des Precompilers ist eine normale COBOL-Quelle, die mit dem COBOL85-Compiler zu übersetzen ist. Zusätzlich extrahiert der Precompiler die SQL-Anweisungen und transformiert sie in sog. SQL-Objekte.

Das übersetzte COBOL-Programm wird mit den SQL-Objekten, den COBOL- und DBMS-Laufzeitmodulen sowie einem Laufzeitsystem für die SQL-Objekte zu einem ausführbaren Programm zusammengebunden.

Dokumentation

SQL/ESQL-Handbücher [15] - [20]

14.5 Universeller Transaktionsmonitor UTM

UTM erlaubt die einfache Erstellung und den Betrieb von Transaktionsanwendungen.

Zur Programmerstellung steht eine genormte Programmschnittstelle (KDCS, DIN 66265) zur Verfügung, die von den meisten Programmiersprachen unterstützt wird.

Zusammen mit dem Formatierungsystem FHS wird die Ein-/Ausgabe über Formate unterstützt.

UTM garantiert, daß eine Transaktion mit allen Datenänderungen entweder vollständig oder nicht durchgeführt wird, und gewährleistet die Konsistenz der Anwenderdaten in Kombination mit UDS, SESAM, LEASY und PRISMA.

UTM bietet Wiederanlauffunktionen bei Anwendungsabbruch, Netzausfall/Netzstörung oder Bildschirmstörungen. UTM unterstützt neben der Dialogverarbeitung auch die Asynchronverarbeitung, wobei der Startzeitpunkt der Programme bestimmt werden kann.

Es wird eine Steuerung zur Aufteilung von Ressourcen (Tasks) angeboten.

Über eine integrierte Steuerung von Druckausgaben auf Remote-Drucker wird ein gesichertes Druckverfahren angeboten.

Durch Teilhaberbetrieb kann eine große Anzahl Terminals mit UTM-Anwendungen arbeiten.

Für Abrechnungszwecke steht ein auf den Teilhaberbetrieb abgestimmtes Accounting-Verfahren zur Verfügung.

UTM bietet umfangreiche Datenschutzmechanismen für den Zugang zu Anwendungen und die Auswahl von Teilfunktionen einer Anwendung.

UTM dient als Basis für eine Reihe von Softwareprodukten.

Dokumentation

UTM-Handbücher [26] - [29]

15 Meldungen des COBOL85-Systems

Der COBOL85-Compiler und das COBOL85-Laufzeitsystem protokollieren umfassend alle Fehler, die während der Übersetzung und beim Ablauf eines COBOL-Programmes auftreten. Die dabei ausgegebenen Meldungen lassen sich in zwei Gruppen einteilen:

1. Meldungen, die sich auf Fehler im COBOL-Quellprogramm beziehen: Sie werden vom Compiler am Ende der Übersetzung in einer Fehlerliste und/oder Fehlerdatei ausgegeben und haben folgenden Aufbau:

```
Msg-Index      Source Seq. No       Severity Code     Error Text
```

   Dabei bezeichnet

Msg-Index	eine fünfstellige (sedezimale) Fehlermeldungsnummer (Die beiden ersten Zeichen geben das Modul des Compilers an, das den Fehler erkannt hat.),
Source Seq. No	die Folgenummer der Quellprogrammzeile, in welcher der Fehler auftritt,
Severity Code	die Fehlerklasse, der der Fehler zuzurechnen ist und
Error Text	den Text der Fehlermeldung. Er enthält eine genauere Beschreibung des Fehlers und zeigt eventuell eine Umgehungsmöglichkeit auf.

   Meldungstexte können wahlweise auf englisch oder deutsch ausgegeben werden; die Sprache läßt sich über das SDF-Kommando
   MODIFY-MSG-ATTRIBUTES TASK-LANGUAGE=E/D auswählen

   Eine aktuelle Liste aller Fehlermeldungen des COBOL85-Compilers kann mit COMOPT PRINT-DIAGNOSTIC-MESSAGES=YES bzw. über die SDF-Option COMPILER-ACTION=PRINT-MESSAGE-LIST angefordert werden (siehe Abschnitte 4.2 bzw. 3.3.4).

Achtung

Werden Fehler gemeldet, deren Text mit SE-1 oder S.E. beginnt, ist in jedem Fall der Systemverwalter/-berater zu verständigen.

Fehlerklasse	Bedeutung	
F	Hinweismeldung	Der Compiler hat im Quellprogramm Sprachmittel erkannt, die – Spracherweiterungen gegenüber der COBOL-Norm ANS85 darstellen, – von künftigen COBOL-Normen nicht mehr unterstützt werden, – gemäß FIPS (Federal Information Processing Standard) einer bestimmten Sprachmenge zuzuordnen sind. COBOL85 gibt Hinweise der Klasse F nur aus,wenn sie explizit mit COMOPT ACTIVATE-WARNING-MECHANISM=YES bzw.ACTIVATE-FLAGGING = ANS85 / FIPS(...) angefordert werden.
I	Hinweismeldung	Der Compiler hat Steueranweisungen oder COBOL-Sprachelemente erkannt, auf die der Anwender zwar aufmerksam gemacht werden soll, die jedoch nicht die Ausgabe einer Warnungs- oder Fehlermeldung rechtfertigen.
0	Warnungsmeldung	Möglicherweise wurde im Quellprogramm ein Fehler gemacht; trotz dieses Fehlers ist der Programmablauf möglich.
1	Fehlermeldung	Der Compiler hat einen Fehler entdeckt. Normalerweise macht der Compiler eine Korrekturannahme; ein Ablauf des Programms zu Testzwecken ist möglich.
2	Schwerwiegender Fehler	In der Regel wird vom Compiler keine Korrekturannahme gemacht; die fehlerhafte Anweisung wird nicht generiert.
3	Abbruchfehler	Es ist ein so schwerwiegender Fehler aufgetreten, daß der Compiler nicht in der Lage ist, die Übersetzung fortzusetzen.

Tabelle 14-1: Fehlerklassen und ihre Bedeutung

2. Meldungen, die
   - der Compiler über den Ablauf und die Beendigung des Übersetzungslaufes generiert (COB90xx),
   - die das COBOL85-Laufzeitsystem über den Ablauf und die Beendigung des Anwenderprogramms erzeugt (COB91xx).
   - Meldungen des POSIX-Treibers für COBOL (COB92xx)

   Sie werden während der Übersetzung bzw. des Programmablaufs nach SYSOUT ausgegeben und haben folgenden Aufbau:

   { COB90nn / COB91nn / COB92nn } Text

   Dabei bezeichnet

COB90nn COB91nn COB92xx	die Kennummer der Meldung
Text	den Text der Meldung. Er enthält – einen Hinweis zum Ablauf des Compilers oder Anwenderprogramms oder – eine genauere Beschreibung des aufgetretenen Fehlers und – in manchen Fällen die Anforderung einer Eingabe durch den Anwender, mit der der Fehler umgangen werden kann. Meldungstexte können wahlweise auf englisch oder deutsch ausgegeben werden; die Sprache läßt sich über das SDF-Kommando MODIFY-MSG-ATTRIBUTES TASK- LANGUAGE=E/D auswählen.

   Der in den Meldungen COB9101 und COB9102 "COMPILATION UNIT ID programmname" genannte Programmname bezeichnet immer ein getrennt übersetztes Programm. Dabei kann es sich um ein einzelnes Programm oder um das äußerste Programm eines geschachtelten Programms handeln.

   Die Angabe COMOPT GENERATE-LINE-NUMBER=YES bzw. ERR-MSG-WITH-LINE-NR=YES in der SDF-Option RUNTIME-OPTIONS bewirkt, daß statt der Meldung COB9101 zu jeder Meldung des Programms die Meldung 9102 ausgegeben wird, die auch die Nummer der Quellprogrammzeile enthält, bei deren Ausführung die Meldung ausgegeben wird.

Die (im Verlauf der Übersetzung ausgegebenen) Meldungen COB9004, COB9017, COB9095, COB9097 und COB9099 werden unterdrückt, wenn vor dem Aufruf des Compilers der Auftragsschalter 4 gesetzt wird.

3. „Alte“ 90xx-Meldungen des COBOL-Laufzeitssystems über den Ablauf von Objekten, die von früheren Compilerversionen (< 2.1B) übersetzt wurden:

alte Meldungsnummer	entspricht neuer Meldungsnummer
9034	entfällt
9054	COB9112
9067	COB9151
9068	COB9168
9069	COB9169
9070	COB9154
9071	COB9171
9074	COB9134
9076	COB9176
9077	COB9117
9079	COB9179

Die folgende Liste stellt die wichtigsten Meldungen des COBOL85-Compilers und des Laufzeitsystems zusammen. Sie enthält für jede Meldung

– Meldungsnummer und Meldungstext (englisch und deutsch) und

– zusätzliche Erläuterungen, die auch über SYSOUT ausgegeben werden; sie sind in folgende Punkte gegliedert

  Typ (der Meldung)

  Bedeutung (der Variablen in der Meldung)

  Verhalten (des Programms)

  Maßnahme (des Anwenders)

Achtung

Bei Auftreten von Meldungen, die in der folgenden Liste nicht aufgeführt sind (Compiler- bzw. Systemfehler), ist generell der Systemberater zu verständigen.

```
COB9000   Copyright (C) FUJITSU TECHNOLOGY SOLUTIONS   2009
                    All rights reserved
COB9000   Copyright (C) FUJITSU TECHNOLOGY SOLUTIONS   2009
                    Alle Rechte vorbehalten
```

Typ
Hinweis

Bedeutung
Copyright

```
COB9001   aaa TOTAL FLAGS: bbb /SI=ccc /S0=ddd /S1=eee /S2=fff /S3=ggg
COB9001   aaa FEHLER GESAMT: bbb /SI=ccc /S0=ddd /S1=eee /S2=fff /S3=ggg
```

Typ
Hinweis

Bedeutung
aaa: Programmname
bbb: Gesamtanzahl der Fehler
ccc: Anzahl der Severity-Code-I-Hinweise
(und ggf. der Severity-Code-F-Hinweise)
ddd: Anzahl der Severity-Code-0-Fehler
eee: Anzahl der Severity-Code-1-Fehler
fff: Anzahl der Severity-Code-2-Fehler
ggg: Anzahl der Severity-Code-3-Fehler

```
COB9002   COMPILATION OF aaa ABORTED
COB9002   DIE UEBERSETZUNG VON aaa WURDE ABGEBROCHEN
```

Typ
Anwenderfehler oder Systemfehler oder COBOL85-Fehler

Bedeutung
aaa: Programmname

Maßnahme
Fehler beheben, nochmal übersetzen;
ggf. Systemverwalter/-berater verständigen

```
COB9004   COMPILATION OF aaa USED bbb CPU SECONDS
COB9004   DIE UEBERSETZUNG VON aaa BENOETIGTE bbb CPU SEKUNDEN
```

Typ
Hinweis

Bedeutung
aaa: Programmname
bbb: Anzahl der Sekunden

```
COB9005   INCORRECT "COMOPT"/"END" STATEMENT OR "END" STATEMENT MISSING
COB9005   DIE "COMOPT"-/"END"-ANWEISUNG IST FALSCH ODER ES FEHLT DIE
          "END"-ANWEISUNG
```

Typ
Anwenderfehler

Verhalten
In Batch/Prozedur: Übersetzung abgebrochen
Im Dialog: weitere COMOPTs angefordert

Maßnahme
"COMOPT"- oder "END"-Anweisung korrigieren bzw. einfügen; nochmals übersetzen

```
COB9006   REASSIGNMENT OF SYSDTA NOT POSSIBLE
COB9006   EINE NEUZUWEISUNG VON SYSDTA IST NICHT MOEGLICH
```

Typ
Systemfehler

Verhalten
Übersetzung abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9008   COMPILER ERROR. OVERLAY=aaa, ADDRESS=bbb, LAST SOURCE SEQUENCE
          NUMBER=ccc
COB9008   COMPILERFEHLER. OVERLAY=aaa ADRESSE=bbb LETZTE QUELLPROGRAMM-
          FOLGENUMMER=ccc
```

Typ
Compilerfehler

Verhalten
Übersetzung abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9017   COMPILATION INITIATED, VERSION IS aaa
COB9017   BEGINN DER UEBERSETZUNG, VERSION aaa
```

Typ
Hinweis

Bedeutung
aaa: Versionsnummer des Compilers

```
COB9027   INPUT TRUNCATED
COB9027   DIE EINGABE WURDE VERKUERZT
```

Typ
Hinweis

Verhalten
Übersetzung läuft weiter

Maßnahme
COMOPT-Zeilen kürzen

```
COB9044  ERROR SIS (&00) OPENING POSIX LISTING-FILE
COB9044  FEHLER SIS (&00) BEIM ÖFFNEN DER POSIX LISTEBDATEI
```

Typ
Anwenderfehler oder Systemfehler

Bedeutung
(&00): SIS-Fehler-Code

Verhalten
Ubersetzung abgebrochen

Maßnahme
Plattenspeicherplatz und Zugriffsrechte überprüfen; ggf. Systemverwalter/-berater verständigen

```
COB9059   COMPILER COBOL85 NOT RELEASED FOR BS2000 VERSIONS LOWER THAN V10.0
COB9059   COMPILER COBOL85 IST NUR FUER BS2000 AB VERSION V10.0 FREIGEGEBEN
```

Typ
Anwenderfehler

Verhalten
Übersetzung abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9085   COBOL85-BC DOES NOT SUPPORT AID
COB9085   COBOL85-BC UNTERSTUETZT AID NICHT
```

Typ
Anwenderfehler

Verhalten
Übersetzung abgebrochen

Maßnahme
Vollausbau des COBOL85 benutzen

```
COB9090   HARDWARE INTERRUPT ADDRESS aaa, OVERLAY=bbb, SOURCE SEQUENCE
          NUMBER=ccc, INTERRUPT WEIGHT CODE=ddd
COB9090   HARDWARE-UNTERBRECHUNG BEI ADRESSE aaa, OVERLAY=bbb, QUELLPROGRAMM-
          FOLGENUMMER=ccc, EREIGNIS-CODE=ddd
```

Typ
Compilerfehler

Bedeutung
aaa: Befehlszähler
bbb: Overlay-Name
ccc: Zeilennummer
ddd: Unterbrechungsgewicht

Verhalten
Übersetzung abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9095   SAVLST FILE aaa CREATED AND CLOSED
COB9095   SAVLST DATEI aaa ERZEUGT UND GESCHLOSSEN
```

Typ
Hinweis

Bedeutung
aaa: Name der erzeugten Listendatei, z.B. SRCLST.COB85.programmname

Verhalten
Übersetzung läuft weiter

```
COB9097   COMPILATION COMPLETED WITHOUT ERRORS
COB9097   DIE UEBERSETZUNG WURDE OHNE FEHLER BEENDET
```

Typ
Hinweis

```
COB9099   aaa
COB9099   aaa
```

Typ
Hinweis

Bedeutung
aaa: COMOPT-Anweisung

Verhalten
Übersetzung läuft weiter

```
COB9100   AWAITING REPLY
COB9100   ANTWORT WIRD ERWARTET
```

Typ
Hinweis

Verhalten
Programmablauf unterbrochen

Maßnahme
Antwort eingeben für "ACCEPT FROM CONSOLE"

```
COB9101   COMPILATION UNIT ID aaa
COB9101   UBERSETZUNGSEINHEIT aaa
```

Typ
Hinweis

Bedeutung
Ergänzung zum Text der vorher ausgegebenen Meldung

```
COB9102   COMPILATION UNIT ID aaa, PROCEDURE DIVISION LINE NUMBER=bbb
COB9102   UBERSETZUNGSEINHEIT aaa, PROCEDURE DIVISION ZEILENNUMMER=bbb
```

Typ
Hinweis

Bedeutung
Ergänzung zum Text der vorher ausgegebenen Meldung

```
COB9105   REPLY T (TERMINATE) OR D (DUMP)
COB9105   BITTE T (TERMINATE) ODER D (DUMP) EINGEBEN
```

Typ
Anwender- oder Systemfehler, der durch die vorhergehende Meldung beschrieben wurde.

Verhalten
Programm wartet auf Antwort

Maßnahme
T für Programmabbruch,
D für Dump und Programmabbruch eingeben

```
COB9108   NESTING TOO DEEP
COB9108   SCHACHTELUNG ZU TIEF
```

Typ
Anwenderfehler

Bedeutung
Ergänzung zur vorher ausgegebenen Meldung

```
COB9109   ALTERNATE-KEYS FOR LINK= aaa DO NOT MATCH THE ALTERNATE-KEYS IN THE
          PROGRAM
COB9109   ALTERNATE-KEYS FUER LINK= aaa STIMMEN NICHT MIT DEN ALTERNATE-KEYS
          IM PROGRAMM UEBEREIN.
```

Typ
Anwenderfehler

Bedeutung
aaa: Linkname der Datei

Verhalten
Programm abgebrochen

Maßnahme
Programm und/oder Datei ändern

```
COB9110   ERROR IN A CREAIX-MACRO: SUBRETURNCODE= aaa  MAINRETURNCODE= bbb
          DMSERRCODE= ccc
COB9110   FEHLER IN EINEM CREAIX-MAKRO: SUBRETURNCODE= aaa,
          MAINRETURNCODE= bbb  DMSERRCODE= ccc
```

Typ
Anwenderfehler oder Systemfehler

Verhalten
Programm abgebrochen

Maßnahme
Abhängig vom Makro-/DVS-Fehlercode korrigieren oder Systemverwalter/-berater verständigen

```
COB9111   ERROR IN A SHOWAIX-MACRO: SUBRETURNCODE= aaa  MAINRETURNCODE= bbb
          DMSERRCODE= ccc
COB9111   FEHLER IN EINEM SHOWAIX-MAKRO: SUBRETURNCODE= aaa
          MAINRETURNCODE= bbb  DMSERRCODE: ccc
```

Typ
Anwenderfehler oder Systemfehler

Verhalten
Programm abgebrochen

Maßnahme
Abhängig vom Makro-/DVS-Fehlercode korrigieren oder Systemverwalter/-berater verständigen

```
COB9112   ERROR OCCURRED TAKING CHECKPOINT. RETURNCODE=aaa, DMS=bbb, LINK=ccc
COB9112   BEIM SCHREIBEN EINES WIEDERANLAUFPUNKTES TRAT EIN FEHLER AUF.
          RETURNCODE=aaa, DMS=bbb, LINK=ccc
```

Typ
Anwenderfehler oder Systemfehler

Bedeutung
aaa: einer der folgenden Werte in Register 15:

- 04 Arbeitsbereich im Klasse-5-Speicher konnte nicht zugewiesen werden.
- 08 Fehler in der Operandenliste des Benutzers.
- 0C Fixpunktdatei ist nicht eröffnet.
- 10 Keine Sekundärzuweisung oder ungültiges Schreiben.
- 14 FCB nicht PAM, oder OPEN nicht INOUT oder OUTIN.
- 18 Nicht-behebbaren Fehler vom DVS erhalten.
- 1C Fehler in der Katalogverwaltung

24 Fixpunkt konnte nicht gesetzt werden, weil ein gemeinsamer Speicherbereich verwendet wurde (Makro ENAMP ist wirksam).
28 Fixpunkt konnte nicht gesetzt werden, weil eine Serialisierungskennung vorhanden ist (Makro ENASI ist wirksam).
2C Fixpunkt konnte nicht gesetzt werden, weil eine Ereigniskennung vorhanden ist (Makro ENAEI ist wirksam).
30 Fixpunkt konnte nicht gesetzt werden, weil ein Contingency-Prozeß definiert wurde (Makro ENACO ist wirksam).

Verhalten
Programm wird fortgesetzt

Maßnahme
Systemverwalter/-berater verständigen

```
COB9117   LINK= aaa NO ENTRY IN CATALOG. ISSUE NEW SET-FILE-LINK COMMAND
COB9117   LINK= aaa KEIN KATALOG EINTRAG. BITTE NEUES SET-FILE-LINK-KOMMANDO
          EINGEBEN
```

Typ
Anwenderfehler

Bedeutung
aaa: Linkname

Verhalten
Programm unterbrochen

Maßnahme
Gültiges SET-FILE-LINK-Kommando und RESUME-PROGRAM eingeben. Bei einem wiederum ungültigen SET-FILE-LINK-Kommando wird die Fehlermeldung nicht wiederholt, und das Programm wird abgebrochen.

```
COB9118   "STOP LITERAL" - AWAITING REPLY aaa
COB9118   "STOP LITERAL" - ANTWORT ERWARTET aaa
```

Typ
Hinweis

Bedeutung
aaa: ausgegebenes Literal

Verhalten
Programm unterbrochen; Meldung an den Bedienplatz

Maßnahme
Operator-Eingabe abwarten; Eingabe des Operators zur Programmfortsetzung ist beliebig

```
COB9119   ABNORMAL TERMINATION. USERS RETURN CODE=aaa, COBOL RETURN CODE=bbb
COB9119   ABNORMALE BEENDIGUNG. ANWENDER RETURN CODE=aaa, COBOL RETURN CODE=bbb
```

Typ
Anwenderfehler oder Systemfehler

Bedeutung
aaa: >0. Vom Programm wurde ein Anwender-Return-Code gesetzt, was zur Programmbeendigung führt.
bbb: >0. Vom COBOL85-System wurde ein Fehler festgestellt und zu Diagnosezwecken ein interner Return-Code gesetzt. Jedem Return-Code ist eine Fehlermeldung oder ein Anwender-Return-Code zugeordnet, aus der/dem seine Bedeutung ersichtlich ist. Falls das Programm von einer Jobvariable überwacht wird, wird außerdem der interne Return-Code in deren Rückkehrcode-Anzeige übernommen (siehe dazu Abschnitt 6.6, Tabelle 6-1, „Rückkehr-Code-Anzeige in Jobvariablen“)

Maßnahme
Vorher ausgegebene Meldung beachten, Programm bzw. Zuweisung ändern, ggf. Systemverwalter/-berater verständigen

```
COB9120   JOB-VARIABLES ARE NOT SUPPORTED IN THIS OPERATING SYSTEM
COB9120   JOB-VARIABLE WERDEN IN DIESEM BETRIEBSSYSTEM NICHT UNTERSTUETZT
```

Typ
Hinweis

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Liefereinheit „Jobvariablen“ anmieten

```
COB9121   END OF FILE ON "ACCEPT" FROM SYSIPT
COB9121   BEI "ACCEPT" VON SYSIPT WURDE DATEIENDE ERKANNT
```

Typ
Hinweis

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Zuweisung von SYSIPT prüfen, ggf. neu zuweisen und Programm erneut starten

```
COB9122   END OF FILE ON "ACCEPT" FROM SYSDTA
COB9122   BEI "ACCEPT" VON SYSDTA WURDE DATEIENDE ERKANNT
```

Typ
Hinweis

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Zuweisung von SYSDTA prüfen, ggf. neu zuweisen und Programm erneut starten.

```
COB9123   FUNCTION aaa IN STATEMENT bbb: UNEXPECTED ERROR
COB9123   FUNKTION aaa IN ANWEISUNG bbb: UNERWARTETER FEHLER
```

Typ
Anwenderfehler oder Systemfehler

Bedeutung
aaa: Name der Standardfunktion
bbb: COBOL-Anweisung

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm korrigieren, ggf. Systemverwalter/-berater verständigen

```
COB9124   FUNCTION aaa IN STATEMENT bbb: NOT ENOUGH ARGUMENTS SPECIFIED
COB9124   FUNKTION aaa IN ANWEISUNG bbb: ES WURDEN NICHT GENUEGEND ARGUMENTE
          ANGEGEBEN
```

Typ
Anwenderfehler

Bedeutung
aaa: Name der Standardfunktion
bbb: COBOL-Anweisung

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm korrigieren

```
COB9125   FUNCTION aaa IN STATEMENT bbb: ARGUMENT HAS INVALID VALUE
COB9125   FUNKTION aaa IN ANWEISUNG bbb: EIN ARGUMENT HAT EINEN UNGUELTIGEN
          WERT
```

Typ
Anwenderfehler

Bedeutung
aaa: Name der Standardfunktion
bbb: COBOL-Anweisung
ccc: Zeilennummer

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm korrigieren

COB9126 FUNCTION aaa IN STATEMENT bbb: ARGUMENT HAS INVALID LENGTH
COB9126 FUNKTION aaa IN ANWEISUNG bbb: EIN ARGUMENT HAT EINE UNGUELTIGE LAENGE

Typ
Anwenderfehler

Bedeutung
aaa: Name der Standardfunktion
bbb: COBOL-Anweisung
ccc: Zeilennummer

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm korrigieren

COB9127 FUNCTION aaa IN STATEMENT bbb: TABLE SUBSCRIPTED WITH "ALL" HAS NO ELEMENT
COB9127 FUNKTION aaa IN ANWEISUNG bbb: EINE MIT "ALL" SUBSKRIBIERTE TABELLE HAT KEINE ELEMENTE

Typ
Anwenderfehler

Bedeutung
aaa: Name der Standardfunktion
bbb: COBOL-Anweisung
ccc: Zeilennummer

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm korrigieren

```
COB9128   ABNORMAL TERMINATION. USERS RETURN CODE=aaa
COB9128   ABNORMALE BEENDIGUNG. ANWENDER RETURN CODE=aaa
```

Typ
Anwenderfehler oder Systemfehler

Bedeutung
aaa: >0. Vom Programm wurde ein Anwender-Return-Code gesetzt, was zur Programmbeendigung führt.

Maßnahme
Programm ändern, ggf. Systemverwalter/-berater verständigen.

```
COB9131   ACCESS TO JOB-VARIABLE aaa FAILED. JOB-VARIABLE IS EMPTY
COB9131   FEHLER BEI ZUGRIFF ZUR JOB-VARIABLEN aaa. JOB-VARIABLE IST LEER
```

Typ
Anwenderfehler

Bedeutung
aaa: Name der Jobvariablen

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Jobvariable vor Ablauf versorgen

```
COB9132   NUMBER OF PARAMETERS IN "CALL" STATEMENT IS NOT EQUAL TO NUMBER OF
          PARAMETERS EXPECTED BY THE CALLED SUBPROGRAM
COB9132   DIE PARAMETERANZAHL EINER "CALL"-ANWEISUNG IST UNGLEICH DER
          PARAMETERANZAHL DES GERUFENEN PROGRAMMS
```

Typ
Hinweis

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
- ggf. auch das aufrufende Programm mit COMOPT CHECK-PARAMETER-COUNT=YES bzw. PROC-ARGUMENT-NR=YES (SDF-Option RUNTIME-CHECKS) übersetzen,
- andernfalls Programm dahingehend ändern, daß die übergebenen mit den erwarteten Parametern anzahlmäßig übereinstimmen

```
COB9133   COBOL85 RUNTIME SYSTEM IN CRTE NOT RELEASED FOR BS2000 VERSIONS
          LOWER THAN V10.0
COB9133   COBOL85 LAUFZEITSYSTEM IM CRTE IST NUR FUER BS2000 AB VERSION V10.0
          FREIGEGEBEN
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9134   A SORT SPECIAL REGISTER HOLDS AN INVALID VALUE
COB9134   EIN SORT-SONDERREGISTER ENTHAELT EINEN UNGUELTIGEN WERT
```

Typ
Hinweis

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm korrigieren

```
COB9140   REFERENCE MODIFICATION: RANGE VIOLATION IN aaa STATEMENT, RELATIVE
          ADDRESS IS bbb, COMPUTED LENGTH IS ccc, SIZE OF DATA ITEM IS ddd
COB9140   REFERENZ-MODIFIZIERUNG: UEBERSCHREITUNG DER DATENFELDGRENZEN BEI
          ANWEISUNG aaa, RELATIVE ADRESSE: bbb, BERECHNETE LAENGE: ccc, LAENGE
          DES DATENFELDES: ddd
```

Typ
Anwenderfehler

Bedeutung
aaa: Anweisung
bbb: Wert der Adressangabe
ccc: Wert der Längenangabe bzw. berechnete Länge, falls keine Länge angegeben wurde
ddd: Länge des teilfeldselektierten Datenfeldes

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm korrigieren

```
COB9142   UNALTERED "GO TO".
COB9142   "GO TO" OHNE "ALTER"
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern: GO TO ohne Paragraphen/Kapitel erst durchlaufen, nachdem ein ALTER dafür ausgeführt wurde

```
COB9143   VOLUME aaa UNEXPIRED PURGE DATE.
COB9143   FUER DEN DATENTRAEGER aaa IST DAS FREIGABE-DATUM NOCH NICHT ERREICHT.
```

Typ
Anwenderfehler

Bedeutung
aaa: Archivnummer des Datenträgers

Verhalten
Programmlauf unterbrochen

Maßnahme
Datenträger verwenden, dessen Freigabedatum bereits erreicht ist.

```
COB9144   SUBSCRIPT-/INDEX-RANGE VIOLATION IN aaa STATEMENT, VALUE OF
          SUBSCRIPT/INDEX IS bbb, TABLE BOUNDARY IS ccc
COB9144   UEBERSCHREITUNG DES SUBSKRIPT-/INDEXBEREICHES BEI ANWEISUNG: aaa,
          SUBSKRIPT-bzw. INDEXWERT: bbb TABELLENGRENZE: ccc
```

Typ
Anwenderfehler

Bedeutung
aaa: Anweisung
bbb: ungültiger Indexwert
ccc: maximal zulässiger Indexwert

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm ändern

```
COB9145   RANGE VIOLATION IN aaa STATEMENT, VALUE OF "DEPENDING ON" ELEMENT
          IS bbb, TABLE BOUNDARY IS ccc
COB9145   UEBERSCHREITUNG DES SUBSKRIPT-/INDEXBEREICHS BEI ANWEISUNG aaa,
          WERT DES "DEPENDING ON"-ELEMENTS: bbb, TABELLENGRENZE: ccc
```

Typ
Anwenderfehler

Bedeutung
aaa: Anweisung
bbb: ungültiger Wert im DEPENDING ON-Feld
ccc: maximal zulässiger Wert

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Programm ändern

```
COB9146   COBOL85 RUNTIME SYSTEM VERSION aaa IN CRTE IS INCOMPATIBLE WITH
          OBJECT CODE COMPILED BY COBOL85 VERSION bbb
COB9146   COBOL85 LAUFZEIT SYSTEM VERSION aaa IM CRTE VERTRAEGT SICH NICHT MIT
          VON COBOL85 VERSION bbb UEBERSETZTEM OBJEKT-CODE
```

Typ
Anwenderfehler

Bedeutung
aaa: Versionsnummer des Laufzeitsystems
bbb: Versionsnummer des Compilers
Das Laufzeitsystem ist älter als der Compiler, der das Programm erzeugte.

Verhalten
Programm abgebrochen

Maßnahme
Verträgliches Laufzeitsystem als gemeinsam benutzbares Laufzeitsystem installieren

```
COB9148   PROGRAM-NAME IN "CALL" OR "CANCEL" STATEMENT VIOLATES SYSTEM
          CONVENTION FOR ESD SYMBOLS
COB9148   DER PROGRAMMNAME IN EINER "CALL"- ODER "CANCEL"-ANWEISUNG ENTSPRICHT
          NICHT DEN KONVENTIONEN FUER ESD SYMBOLE
```

Typ
Anwenderfehler

Verhalten
CALL ohne EXCEPTION-Klausel: Programmabbruch
CALL mit EXCEPTION-Klausel: Fortsetzung mit EXCEPTION-Klausel
CANCEL: Fortsetzung mit Anweisung nach CANCEL

Maßnahme
Programmnamen richtig schreiben

```
COB9149   INCOMPATIBLE DATA IN NUMERIC EDITED ITEM
COB9149   IN NUMERISCH EDITIERTEM DATENFELD SIND INKOMPATIBLE DATEN
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9151   PC=aaa, STATUS=bbb, FILE=ccc, LINK=ddd, DMS/SIS=eee, NO USE ERROR
          PROCEDURE
COB9151   PC=aaa, STATUS=bbb, DATEI=ccc, LINK=ddd, DMS/SIS=eee, KEINE USE
          PROZEDUR
```

Typ
Anwenderfehler oder Systemfehler: schwerwiegender Fehler bei Ein-/Ausgabe oder Programmierfehler (keine USE-Prozedur, kein INVALID KEY oder keine AT END-Klausel)

Bedeutung
aaa: aktueller Stand des Befehlszählers
bbb: aktueller COBOL FILE-STATUS
ccc: Dateiname
ddd: Linkname
eee: DVS-Fehlercode

Verhalten
Programm abgebrochen

Maßnahme
abhängig vom DVS-Fehler-Code korrigieren oder Systemverwalter/-berater verständigen

```
COB9152   NO CONNECTION WITH DATABASE DURING PROGRAM INITIALIZATION
COB9152   WAEHREND DER PROGRAMMINITIALISIERUNG KONNTE KEINE VERBINDUNG ZUR
          DATENBANK HERGESTELLT WERDEN
```

Typ
Anwenderfehler (entweder DBH nicht geladen oder SET-FILE-LINK-Kommando inkorrekt)

Verhalten
Programm abgebrochen

Maßnahme
DBH laden und Programm mit evtl. geändertem SET-FILE-LINK-Kommando neu starten

```
COB9154   REPORT DEFINED AT LINE aaa: "INITIATE" STATEMENT ISSUED TO REPORT
          WHICH IS NOT TERMINATED
COB9154   REPORT DEFINIERT IN ZEILE aaa: EINE "INITIATE"-ANWEISUNG SOLL
          AUSGEFUEHRT WERDEN, OBWOHL FUER DEN REPORT NOCH KEINE "TERMINATE"-
          ANWEISUNG GEGEBEN WURDE.
```

Typ
Anwenderfehler

Bedeutung
aaa: Zeilennummer

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9155   ERROR ON EXIT FROM THE USE-PROCEDURE ON PC aaa
COB9155   FEHLER BEIM VERLASSEN DER USE-PROZEDUR AUF PC aaa
```

Typ
Anwenderfehler

Bedeutung
aaa: Befehlszähler

Verhalten
Programm abgebrochen

Maßnahme
Programm korrigieren

```
COB9156   SUB-SCHEMA MODULE TOO SMALL TO PROCESS AN EXTENSIVE DML-STATEMENT
COB9156   DER SUB-SCHEMA MODUL IST ZU KLEIN, UM EIN UMFANGREICHES DML-STATEMENT
          ZU VERARBEITEN
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern; kürzere DML-Anweisungen, da FIND-7, FETCH-7 zu umfangreich

```
COB9157   PROGRAM REFERENCED BY "CALL" OR "CANCEL" STATEMENT IS STILL ACTIVE
COB9157   DAS MIT EINER "CALL"- ODER "CANCEL"-ANWEISUNG ANGESPROCHENE
          PROGRAMM IST NOCH AKTIV
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programmaufrufe überprüfen und ändern (keine Rekursion zulässig)

```
COB9158   MORE THAN 9 RECURSIVE CALLS OF DEPENDING PARAGRAPHS
COB9158   MEHR ALS 9 REKURSIVE AUFRUFE VON DEPENDING PARAGRAPHEN
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9160   PROGRAMS COMPILED BY COBOL85 VERSIONS LOWER THAN V2.0A FOUND IN
          RUNUNIT USING "INITIAL" OR "CANCEL"
COB9160   DIE RUNUNIT ENTHAELT MIT COBOL85 VERSIONEN KLEINER V2.0A UEBERSETZTE
          PROGRAMME, OBWOHL "INITIAL" ODER "CANCEL" VERWENDET WIRD
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Alte Programme mit neuer COBOL85-Version übersetzen

```
COB9162   INCONSISTENT DESCRIPTIONS FOR EXTERNAL FILE aaa IN DIFFERENT PROGRAMS
COB9162   DIE EXTERNE DATEI aaa WURDE IN VERSCHIEDENEN PROGRAMMEN
          UNTERSCHIEDLICH BESCHRIEBEN
```

Typ
Anwenderfehler

Bedeutung
aaa: Name der externen Datei

Verhalten
Programm abgebrochen

Maßnahme
In allen Programmen die gleiche Beschreibung für die externe Datei verwenden

```
COB9163   NOT ENOUGH SPACE AVAILABLE FOR DYNAMIC DATA OR FUNCTION IDENTIFIER
COB9163   ES IST NICHT GENUEGEND SPEICHER FUER DYNAMIC DATEN ODER
          FUNKTIONSBEZEICHNER VORHANDEN
```

Typ
Anwenderfehler oder Systemfehler

Verhalten
Programm abgebrochen

Maßnahme
DYNAMIC-Daten im Programm kleiner machen oder Länge und Anzahl der Argumente in nichtnumerischen Funktionen verringern oder ggf. ADDRSPACE im Join-Eintrag erhöhen

```
COB9164   PROGRAM aaa REFERENCED BY "CALL IDENTIFIER" STATEMENT CAN NOT BE MADE
          AVAILABLE, BLS RETURNCODE= bbb
COB9164   DAS MIT EINER "CALL IDENTIFIER"-ANWEISUNG ANGESPROCHENE PROGRAMM aaa
          KANN NICHT VERFUEGBAR GEMACHT WERDEN, RETURNCODE VON BLS= bbb
```

Typ
Anwenderfehler

Verhalten
Programmlauf abgebrochen

Bedeutung
aaa: Name des nachzuladenden Programms
bbb: Returncode des BIND/LINK-Makros

Maßnahme
Für das Programm die Bibliothek mit dem Linknamen COBOBJCT zuweisen;
Für das LZS die Bibliothek mit dem Linknamen BLSLIBxx zuweisen

```
COB9168   REPORT DEFINED IN LINE aaa: GROUP bbb REQUIRES TOO MANY LINES
COB9168   REPORT DEFINIERT IN ZEILE aaa: GRUPPE bbb ENTHAELT ZU VIELE ZEILEN
```

Typ
Anwenderfehler

Bedeutung
aaa: Zeilennummer
bbb: Name der Gruppe

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9169   REPORT DEFINED IN LINE aaa: GROUP bbb LINE CONFLICTS WITH HEADING
COB9169   REPORT DEFINIERT IN ZEILE aaa: GRUPPE bbb EINE ZEILE STEHT IM
          WIDERSPRUCH ZUR SEITENKOPFBEGRENZUNG
```

Typ
Anwenderfehler

Bedeutung
aaa: Zeilennummer
bbb: Name der Gruppe

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9171   REPORT DEFINED IN LINE aaa: "GENERATE" ISSUED TO TERMINATED REPORT
COB9171   REPORT DEFINIERT IN aaa: FUER EINEN BEREITS ABGESCHLOSSENEN REPORT
          WURDE EINE "GENERATE"-ANWEISUNG GEGEBEN
```

Typ
Anwenderfehler

Bedeutung
aaa: Zeilennummer

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9173   SORT NO. aaa UNSUCCESSFUL
COB9173   SORTLAUF MIT NR. aaa NICHT ERFOLGREICH
```

Typ
Anwenderfehler oder Systemfehler

Bedeutung
aaa: Nummer des SORT-Laufs

Verhalten
Programm abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9174   DML-EXCEPTION ON STATEMENT PC aaa, DB-STATUS=bbb - ccc. EXCEPTION
COB9174   DML-SONDERZUSTAND BEI ANWEISUNG AUF PC aaa, DB-STATUS=bbb - ccc.
          SONDERZUSTAND
```

Typ
Anwenderfehler

Bedeutung
aaa: Befehlszähler
bbb: DB-Statuswert
ccc: Zählung der Ausnahmebedingungen

Verhalten
Programm abgebrochen

Maßnahme
Programm prüfen, ggf. ändern

```
COB9175   DMS/SIS-EXCEPTION ON STATEMENT PC aaa, FILE-STATUS=bbb (DMS/SIS=ccc)
          ON ddd - eee. EXCEPTION
COB9175   DMS/SIS-SONDERZUSTAND BEI ANWEISUNG AUF PC aaa, FILE-STATUS=bbb
          (DMS/SIS=ccc) ON ddd - eee. SONDERZUSTAND
```

Typ
Anwenderfehler

Bedeutung
aaa: Befehlszähler
bbb: File-Statuswert
ccc: DVS/SIS-Fehlercode
ddd: Linkname
eee: Zählung der Ausnahmebedingungen

Verhalten
Programm abgebrochen

Maßnahme
Programm prüfen, ggf. ändern

```
COB9176   REPORT DEFINED IN LINE aaa: "TERMINATE" ISSUED TO REPORT WHICH IS
          NOT INITIATED
COB9176   REPORT DEFINIERT IN ZEILE aaa: EINE "TERMINATE"-ANWEISUNG WURDE
          DURCHGEFUEHRT OBWOHL NOCH KEINE "INITIATE"-ANWEISUNG GEGEBEN WURDE
```

Typ
Anwenderfehler

Bedeutung
aaa: Zeilennummer

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9178   THE LENGTH OF THE RECORD TO RELEASE TO THE SORT IS LARGER / LESS
          THAN THE MAXIMUM / MINIMUM RECORD LENGTH OF THE SORT FILE
COB9178   DIE LAENGE DES AN DEN SORT ZU UEBERGEBENDEN SATZES IST GROESSER /
          KLEINER ALS DER MAXIMALE / MINIMALE SATZ DER SORT-DATEI
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern, Satzlängen angleichen

```
COB9179   THE LENGTH OF THE RECORD RETURNED FROM SORT IS LARGER / LESS THAN
          THE MAXIMUM / MINIMUM RECORD LENGTH OF THE GIVING FILE
COB9179   DIE LAENGE DES SORTIERTEN SATZES IST GROESSER / KLEINER ALS DIE
          MAXIMALE / MINIMALE SATZLAENGE DER GIVING-DATEI.
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern, Satzlängen angleichen

```
COB9180   "RELEASE"/"RETURN" OUTSIDE "SORT"/"MERGE" CONTROL
COB9180   "RELEASE"/"RETURN" AUSSERHALB DER "SORT"/"MERGE"-STEUERUNG
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9181   THE DATABASE-HANDLER HAS NOT YET PROCESSED THE LAST DML-STATEMENT
COB9181   DER DATABASE-HANDLER HAT DAS LETZTE DML-STATEMENT NOCH NICHT
          ABGEARBEITET
```

Typ
Anwenderfehler
Der DBH ist durch STXIT unterbrochen und bekommt eine neue DML-Anweisung, ehe die unterbrechende Anweisung abgearbeitet werden konnte.

Verhalten
Programm abgebrochen

Maßnahme
Programm korrigieren

```
COB9184   "SORT" INSIDE "SORT" CONTROL
COB9184   "SORT" INNERHALB DER "SORT"-STEUERUNG
```

Typ
Anwenderfehler

Verhalten
Programm abgebrochen

Maßnahme
Programm ändern

```
COB9192   END OF PROCEDURE DIVISION OR ROOT SEGMENT OF MAINPROGRAM
          ENCOUNTERED WITHOUT "STOP RUN" HAVING BEEN EXECUTED
COB9192   DAS ENDE DER PROCEDURE DIVISION BZW. DES ROOT-SEGMENTS IM
          HAUPTPROGRAMM WURDE ERREICHT OHNE DASS "STOP RUN" AUSGEFUEHRT WURDE
```

Typ
Hinweis

Verhalten
Programm abgebrochen

Maßnahme
An das logische Ende des Hauptprogramms eine "STOP RUN"-Anweisung setzen.

```
COB9193   UNRECOVERABLE ERROR DURING DISPLAY UPON TERMINAL
COB9193   NICHT BEHEBBARER FEHLER WAEHREND EINER AUSGABE AUF TERMINAL
```

Typ
Systemfehler

Verhalten
Programm abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9194   UNRECOVERABLE ERROR ON "ACCEPT" FROM SYSDTA
COB9194   NICHT BEHEBBARER FEHLER BEI "ACCEPT" VON SYSDTA
```

Typ
Systemfehler

Verhalten
Programm abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9195   UNRECOVERABLE ERROR DURING "DISPLAY" UPON SYSLST
COB9195   NICHT BEHEBBARER FEHLER BEI "DISPLAY" AUF SYSLST
```

Typ
Systemfehler

Verhalten
Programm abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9196   ERROR ON INTERFACE RUN-TIME-SYSTEM - OPERATING-SYSTEM IN "ACCEPT"
          OR "DISPLAY" STATEMENT
COB9196   FEHLER AN DER SCHNITTSTELLE LAUFZEITSYSTEM - BETRIEBSSYSTEM IN
          "ACCEPT" ODER "DISPLAY"-ANWEISUNG
```

Typ
Systemfehler

Verhalten
Programm abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9197   ACCESS TO JOB-VARIABLE aaa FAILED. ERROR CODE=bbb
COB9197   FEHLERHAFTER ZUGRIFF ZUR JOB-VARIABLEN aaa FEHLER-CODE=bbb
```

Typ
Hinweis

Bedeutung
aaa: Linkname der JV
bbb: Code der Systemmeldung

Verhalten
Abhängig von COMOPT CONTINUE-AFTER-MESSAGE bzw. ERROR-REACTION (SDF-Option RUNTIME-OPTIONS) wird das Programm fortgesetzt oder abgebrochen

Maßnahme
Zugriffsberechtigungen zu Jobvariablen ändern

```
COB9198   INTERRUPT-CODE=aaa, AT PC=bbb
COB9198   UNTERBRECHUNGS-CODE=aaa, BEI PC=bbb
```

Typ
Hardwareunterbrechung im Anwenderprogramm

Bedeutung
aaa: Unterbrechungsgewicht
bbb: Befehlszähler

Verhalten
Programm abgebrochen

Maßnahme
Programm korrigieren

```
COB9201   POSIX DRIVER FOR COBOL -- VERSION AAA
COB9201   POSIX TREIBER FUER COBOL -- VERSION AAA
```

Typ
Hinweis

Bedeutung
aaa: Versionsnummer des Treibers

```
COB9205   USAGE: COBOL [OPTIONS] FILENAME ...
COB9205   SYNTAX: COBOL [OPTIONEN] DATEINAME ...
```

Typ
Anwenderfehler

Bedeutung
Es wurde keine Datei zur Bearbeitung angegeben

```
COB9206   MISSING ARGUMENT FOR OPTION -XXX
COB9206   FEHLENDES ARGUMENT FUER OPTION -XXX
```

Typ
Anwenderfehler

Bedeutung
-xxx: Option, zu der das Argument fehlt

```
COB9207   WARNING: OPTION XXX IGNORED
COB9207   WARNUNG: OPTION XXX IGNORIERT
```

Typ
Hinweis

Bedeutung
Option xxx wurde nicht benötigt

```
COB9211   COBOL COMPILER RETURNED WITH ERROR NN
COB9211   COBOL COMPILER KEHRT MIT FEHLER NN ZURUECK
```

Typ
Systemfehler

Bedeutung
Programm abgebrochen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9212   CCC: COMMAND NOT FOUND
COB9212   CCC: KOMMANDO NICHT GEFUNDEN
```

Typ
Anwenderfehler oder Systemfehler

Maßnahme
Herausfinden, warum das Kommando nicht gefunden wurde, bzw. Systemverwalter/-berater verständigen

```
COB9213   CANNOT EXECUTE CCC (ERRNO=NN)
COB9213   CCC KANN NICHT AUSGEFÜHRT WERDEN (ERRNO=NN)
```

Typ
Systemfehler

Bedeutung
ccc: Nicht ausfuehrbares Kommando
nn: Nummer des Fehlers, der beim Versuch, das Kommando auszuführen, aufgetreten ist

Maßnahme
Systemverwalter/-berater verständigen

```
COB9214   CANNOT CREATE FILE AAA (ERRNO=NN)
COB9214   DATEI AAA KANN NICHT ERZEUGT WERDEN (ERRNO=NN)
```

Typ
Systemfehler

Maßnahme
aaa: Dateiname
nn: Nummer des Fehlers, der beim Versuch, die Datei anzulegen, aufgetreten ist

Maßnahme
Systemverwalter/-berater verständigen

```
COB9215   CANNOT REMOVE FILE AAA (ERRNO=NN)
COB9215   DATEI AAA KANN NICHT GELOESCHT WERDEN (ERRNO=NN)
```

Typ
Systemfehler

Bedeutung
aaa: Dateiname
nn: Nummer des Fehlers, der beim Versuch, die Datei zu löschen, aufgetreten ist

Maßnahme
Systemverwalter/-berater verständigen

```
COB9216   CANNOT SET ENVIRONMENT VARIABLE XXX
COB9216   UMGEBUNGSVARIABLE XXX KANN NICHT GESETZT WERDEN
```

Typ
Systemfehler

Maßnahme
Systemverwalter/-berater verständigen

```
COB9217   CANNOT GENERATE TEMPORARY FILENAME
COB9217   TEMPORAERER DATEINAME KANN NICHT ERZEUGT WERDEN
```

Typ
Systemfehler

Maßnahme
Systemverwalter/-berater verständigen

```
COB9221   INSUFFICIENT MEMORY
COB9221   SPEICHERMANGEL
```

Typ
Systemfehler

Bedeutung
Das Betriebssystem konnte keinen dynamischen Speicher mehr zur Verfügung stellen

Maßnahme
Systemverwalter/-berater verständigen

```
COB9231   CCC TERMINATED BY SIGNAL NN
COB9231   CCC BEENDET DURCH SIGNAL NN
```

Typ
Hinweis

Bedeutung
Der Prozeß, innerhalb dessen das Kommando ccc ausgeführt wurde, wurde infolge eines Signal mit dem Wert nn abgebrochen

```
COB9232   CCC STOPPED BY SIGNAL NN
COB9232   CCC ANGEHALTEN DURCH SIGNAL NN
```

Typ
Hinweis

Bedeutung
Der Prozeß, innerhalb dessen das Kommando ccc ausgeführt wurde, wurde infolge eines Signals mit dem Wert nn angehalten

```
COB9233   CCC: STRANGE TERMINATION STATUS
COB9233   CCC: UNERWARTETER BEENDIGUNGSSTATUS
```

Typ
Systemfehler

Bedeutung
ccc: Kommando, das sich auf unerwartete Weise beendet hat

Maßnahme
Systemverwalter/-berater verständigen

```
COB9241   CANNOT FORK (ERRNO=NN)
COB9241   FORK() KANN NICHT AUSGEFUEHRT WERDEN (ERRNO=NN)
```

Typ
Systemfehler

Bedeutung
Es konnte kein Sohnprozeß erzeugt werden, als Fehlernummer wurde nn zurückgegeben

Maßnahme
Systemverwalter/-berater verständigen

```
COB9242   WAIT() CALL FAILED (ERRNO=NN)
COB9242   WAIT() MELDET FEHLER (ERRNO=NN)
```

Typ
Systemfehler

Bedeutung
nn: Nummer des Fehlers, der beim wait()-Aufruf aufgetreten ist

Maßnahme
Systemverwalter/-berater verständigen

```
COB9243   UNEXPECTED SON PROCESS RETURNED (PID=NN)
COB9243   UNERWARTETER SOHNPROZEß IST ZURÜCKGEKEHRT (PID=NN)
```

Typ
Systemfehler

Bedeutung
nn: Nummer des Sohnprozesses

Maßnahme
Systemverwalter/-berater verständigen

16 Anhang

16.1 Aufbau des COBOL85-Systems

Das COBOL85-System besteht aus den Modulen des Compilers und den Laufzeitmodulen. Auf die Struktur des Compilers und die Namen der Module wird im folgenden näher eingegangen. Die Laufzeitmodule für COBOL85 sind im Common Runtime Environment (CRTE) enthalten (siehe [2]).

Aufbau des COBOL85-Compilers

Der COBOL85-Compiler besteht aus einer Anzahl von Modulen, die linear gebunden sind.

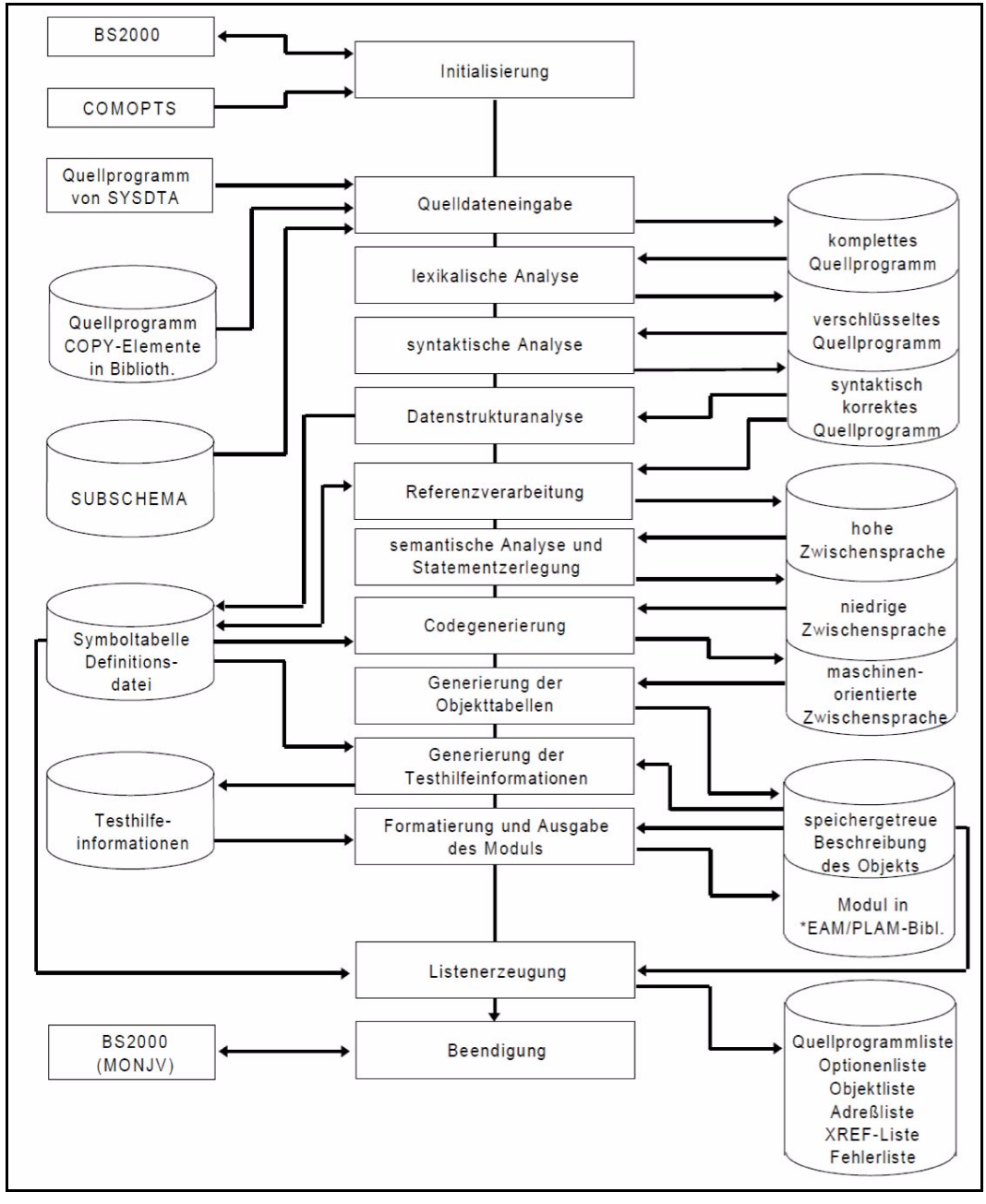
Die einzelnen Module bilden Funktionseinheiten, die durch den Ablauf einer COBOL-Übersetzung und durch die Einteilung eines COBOL-Programms in die einzelnen DIVISIONS vorgegeben werden.

Man kann den Übersetzungsvorgang in folgende Funktionseinheiten gliedern:

1. Initialisierung
2. Quelldateneingabe
3. Lexikalische Analyse
4. Syntaktische Analyse
5. Semantische Analyse
6. Codegenerierung
7. Assemblierungslauf
8. Modulgenerierung
9. Listenerzeugung

Der Aufbau des Compilers und die Anordnung der einzelnen Funktionseinheiten im Arbeitsspeicher ist in folgender Abbildung wiedergegeben.

Aufbau des Compilers



Das COBOL85-Laufzeitsystem

Das COBOL85-Laufzeitsystem ist Bestandteil des **C**ommon **R**un**T**ime **E**nvironment (CRTE), der gemeinsamen Laufzeitumgebung für COBOL85- und C/C++-Programme.

Das CRTE ist in einem eigenen Benutzerhandbuch [2] beschrieben.

Die COBOL85-Laufzeitroutinen stellen dem COBOL85-Compiler bekannte Unterprogramme dar. Sie können im wesentlichen in zwei Gruppen unterteilt werden:

1. Unterprogramme für komplexe COBOL-Anweisungen

Beispiele für komplexe COBOL-Anweisungen sind Handbuch [1] zu entnehmen; aber auch scheinbar einfache Anweisungen (wie z.B. COMPUTE A = B ** C), für die keine entsprechenden Maschinenbefehle existieren, werden durch Bildung von Unterprogrammen und Auslagerung dieser Unterprogramme in vorübersetzte Module realisiert.

2. Unterprogramme zum Anschluß des generierten Moduls an Betriebssystemfunktionen

Diese Unterprogramme dienen hauptsächlich dazu, die Codegenerierung des Compilers völlig betriebssystemunabhängig zu halten. Die dabei möglicherweise auftretenden Effizienzverluste werden weitgehend durch die größere Betriebssystemunabhängigkeit ausgeglichen. Bei Änderung der Schnittstellen zum Betriebssystem genügt im allgemeinen das erneute Binden der vorhandenen Module mit dem neuen Laufzeitsystem.

Wesentliche Funktionen unter diesem Titel sind:

– Anschluß der COBOL-Programme an das Ein-/Ausgabesystem

– Anschluß der COBOL-Programme an SORT

– Anschluß der COBOL-Programme an UDS

– Anschluß der COBOL-Programme an Ablaufteil-Funktionen

In folgender Tabelle sind die Namen und Funktionen der COBOL85-Laufzeitmodule aufgeführt. In der Tabelle nicht enthalten sind diejenigen Laufzeitmodule, die nur aus Kompatibilitätsgründen noch im COBOL85-Laufzeitsystem vorhanden sein müssen, sowie diejenigen Module, die für Zugriffe auf das POSIX-Dateisystem verwendet werden.

Name	Funktion
ITCMAID1 *)	AID-Anschlußmodul (Datenteil)
ITCMECE1 *)	ENTRY, CANCEL, EXIT
ITCMERF1 *)	Fehleranalyseroutine für Ein-/Ausgabe
ITCMINIT *)	ILCS-Initialisierung
ITCMMAT1 *)	Daten für mathematische (IML...-) Funktionen
ITCMMDP0 *)	OCCURS DEPENDING (rekursiv)
ITCMPOVH *)	COBOL85 Programm-Manager
ITCMSMG0 *)	SORT-/MERGE-Anweisung
ITCMTBS1 *)	Tabellensortieren (non sharable)
ITCMTBS2 *)	Tabellensortieren (Daten)
ITCRACA0	ACCEPT-Anweisung
ITCRACX0	ACCEPT-Anweisung für Umgebungsvariable/Kommandozeile
ITCRAID2	AID-Anschlußmodul (Prozedurteil)
ITCRBEG0	Programmsystem-Initialisierungsroutine
ITCRCCL1	CLOSE für INITIAL / CANCEL
ITCRCHP0	RERUN-Klausel mit Angabe ganzzahl RECORDS
ITCRCHP2	RERUN-Klausel für SORT-Dateien und END OF REEL
ITCRCLA0	Vergleich ALL literal
ITCRCLI0	CLOSE-Anweisung für indizierte Dateien
ITCRCLL0	CLOSE-Anweisung für zeilensequentielle Dateien
ITCRCLR0	CLOSE-Anweisung für relative Dateien
ITCRCLS0	CLOSE-Anweisung für sequentielle Dateien
ITCRCVB0	Umwandlung gepackt dezimal nach binär > 15 Stellen
ITCRCVD0	Umwandlung binär nach gepackt dezimal > 15 Stellen
ITCRCVF0	Umwandlung von und nach Gleitpunkt
ITCRDBL0	Verbindungsmodul zum Datenbanksystem UDS
ITCRDFE0	Division extern Gleitpunkt
ITCRDPL0	Division von Dezimalzahlen > 15 Stellen
ITCRDSA0	DISPLAY-Anweisung
ITCRDSI1	Speicherzuweisung DYNAMIC-Daten
ITCRDSX0	DISPLAY-Anweisung für Umgebungsvariable

Name	Funktion
ITCRDYF1	Beschaffung von Speicherplatz für die Funktionen REVERSE, UPPER-CASE, LOWER-CASE
ITCRECE0	ENTRY, CANCEL, EXIT für getrennt übersetzte Programme
ITCREND0	Programmbeendigungsroutine (normal und abnormal)
ITCREV0	Ereignisbehandlung (Rückkehr aus einem fremdsprachigen Unterprogramm)
ITCREV1	Ereignisbehandlung ("recoverable interrupts")
ITCREV2	Ereignisbehandlung ("unrecoverable interrupts")
ITCREV3	Ereignisbehandlung (übrige Ereignisse)
ITCRFAC0	Tabelle für FACTORIAL-Funktion
ITCRFCH0	Meldungsausgabe der Funktionsargumentprüfung
ITCRFCT1	Gleitpunkt-Konstanten
ITCRFDT0	Datumskonvertierungsfunktionen
ITCRFMD0	Funktion MEDIAN
ITCRFMX0	Funktionen MAX, MIN, ORD-MAX, ORD-MIN, RANGE, MIDRANGE
ITCRFNM0	Funktionen NUMVAL, NUMVAL-C
ITCRFPV0	Funktion PRESENT-VALUE
ITCRFRN0	Funktion RANDOM
ITCRFST0	Funktionen REVERSE, UPPER-CASE, LOWER-CASE
ITCRFVR0	Funktion VARIANCE
ITCRHSW0	Setzen und Prüfen von Auftrags-/Benutzerschaltern
ITCRIFA0	FCB-Initialisierung; Steuerroutine
ITCRIFC1	RERUN-Klausel FCB-Generierung
ITCRIFI1	ISAM-FCB-Generierung für indizierte Dateien
ITCRIFL1	SAM-FCB-Generierung für zeilensequentielle Dateien
ITCRIFR1	ISAM-FCB-Generierung für relative Dateien
ITCRIFS1	SAM-FCB-Generierung für sequentielle Dateien
ITCRINI	ILCS/COBOL-Initialisierung
ITCRINI0	INITIALIZE-Anweisung
ITCRINS0	INSPECT-Anweisung
ITCRLHS2	Benutzerkennsatzbehandlung für sequentielle Dateien
ITCRLNL1	LINAGE-Klausel bei WRITE für zeilensequentielle Dateien

Name	Funktion
ITCRLNS1	LINAGE-Klausel bei WRITE für satzsequentielle Dateien
ITCRMAT0	Verbindungsmodul zu den mathematischen (IML...-) Funktionen
ITCRMEV2	Unterbrechungsmeldung für Event-Handling-Routine
ITCRMPL0	Multiplikation von Dezimalzahlen > 15 Stellen
ITCRMSG3	Ausgabe von Fehlermeldungen
ITCRMVE0	MOVE für numerisch-druckaufbereitete Felder
ITCRNED0	Deeditierender MOVE
ITCRNSP0	CALL, CANCEL, ENTRY, EXIT im geschachtelten Programm
ITCROPI0	OPEN-Anweisung für indizierte Dateien
ITCROPL0	OPEN-Anweisung für zeilensequentielle Dateien
ITCROPR0	OPEN-Anweisung für relative Dateien
ITCROPS0	OPEN-Anweisung für sequentielle Dateien
ITCRPAM1	physische Lese-/Schreibroutine für relative Dateien (PAM)
ITCRPCA0	Vergleiche unter PROGRAM COLLATING SEQUENCE
ITCRPCS0	Vergleiche unter PROGRAM COLLATING SEQUENCE
ITCRPND0	ILCS-Routine für UTM-Beendigung
ITCRRCH0	Überprüfung von Tabellengrenzen
ITCRRDI0	READ-/START-Anweisung für indizierte Dateien
ITCRRDL0	READ-/START-Anweisung für zeilensequentielle Dateien
ITCRRDR0	READ-/START-Anweisung für relative Dateien
ITCRRDS0	READ-Anweisung für sequentielle Dateien
ITCRRPW0	REPORT-WRITER-Steuermodul
ITCRSCH0	SEARCH-ALL-Anweisung
ITCRSEG0	Ansprung segmentierter COBOL-Programme
ITCRSPC0	Druckersteuerzeichen-Auswertung
ITCRST11	CODE SET-Tabelle für ASCII
ITCRST21	CODE SET-Tabelle für ISO-7
ITCRSTG0	STRING-Anweisung
ITCRSTP0	STOP literal-Anweisung
ITCRTBS0	Tabellensortieren
ITCRTCA1	Klassentest-Tabelle für Test auf ALPHABETIC

Name	Funktion
ITCRTCD1	Klassentest-Tabelle für Test auf NUMERIC (COMP-3 mit Vorz.)
ITCRTCE1	Klassentest-Tabelle für Test auf NUMERIC (COMP-3 ohne Vorz.)
ITCRTCL1	Klassentest-Tabelle für Test auf ALPHABETIC-LOWER
ITCRTCP1	Klassentest-Tabelle für Test auf ALPHABETIC-UPPER
ITCRTCS1	Klassentest-Tabelle für Test auf NUMERIC (mit Vorzeichen)
ITCRTCU1	Klassentest-Tabelle für Test auf NUMERIC
ITCRTCV0	Klassentest bei Datenfeldern > 256 byte oder Variablen
ITCRUPC2	Steuermodul für Prozedurvereinbarungen
ITCRUST0	UNSTRING-Anweisung
ITCRVCL0	Vergleich für Felder variabler Länge/Adresse oder > 256 byte
ITCRVMA0	MOVE ALL literal
ITCRVMP0	Auffüllen für Felder > 256 byte bei MOVE
ITCRVMV0	MOVE für Felder variabler Länge/Adresse oder > 256 byte
ITCRWRI0	WRITE-/REWRITE-Anweisung für indizierte Dateien
ITCRWRL0	WRITE-/REWRITE-Anweisung für zeilensequentielle Dateien
ITCRWRR0	WRITE-/REWRITE-Anweisung für relative Dateien
ITCRWRS0	WRITE-Anweisung für sequentielle Dateien
ITCRXIT0	FILE STATUS und Fehlerbehandlungsroutine
ITCRXPF0	Potenzierung

*) Modul nicht gemeinsam benutzbar

16.2 Datenbankbedienung (UDS)

In COBOL85-BC nicht unterstützt !

Eine Beschreibung des universellen Datenbanksystems UDS findet sich in den Handbüchern Entwerfen und Definieren [12], Aufbauen und Umstrukturieren [13], Anwendungen programmieren [14].

UDS-Datenbanken werden von Anwenderprogrammen bedient über

– COBOL-DML-Sprachelemente (DML ist integraler Bestandteil von COBOL)

– CALL DML (Datenbankbehandlung über Unterprogrammaufruf).

Der folgende Text beschränkt sich auf COBOL-DML. Ferner wird davon ausgegangen, daß Schema und Subschema bereits generiert sind. Hier werden einzelne Schritte zur Erzeugung eines UDS-Anwenderprogramms kurz dargestellt.

Der Database Handler (DBH) als Kernkomponente des UDS-Datenbanksystems ist zuständig für die Kommunikation zwischen dem Anwenderprogramm und der Datenbank (über das Subschema). Man unterscheidet:

– Linked-in DBH: Er wird in das Anwenderprogramm eingebunden, eignet sich also für den Fall, daß nur ein Anwenderprogramm mit der Datenbank arbeiten soll.

– Independent DBH: Er wird nicht mit in das Anwenderprogramm eingebunden, d.h. er kann mehr als ein Anwenderprogramm steuern (eigener Prozeß).

Aufbau eines COBOL-DML-Programms

```
DATA DIVISION.
    .
    .
    .
SUB-SCHEMA SECTION.
    DB subschema-name WITHIN schema-name.
PROCEDURE DIVISION.
    .
    .
    Folge von COBOL-DML-Anweisungen
    ...    .
```

Die Formate der COBOL-DML-Anweisungen sind in [14] beschrieben.

schema-name/subschema-name werden bei der Schema- bzw. Subschema-Generierung festgelegt.

Übersetzen eines COBOL-DML-Programms

Der COBOL85-Compiler erzeugt aus einem COBOL-DML-Programm ein Programm-Modul und ein Subschema-Modul.

Mittels eines SET-FILE-LINK-Kommandos (mit LINK=DATABASE) wird dem Compiler der Name der Datenbank (dbname) mitgeteilt. Dieser Name wurde schon bei der Datenbank-Generierung verwendet. Mit seiner Hilfe erkennt der Compiler die Datei dbname.COSSD, aus der er das Subschema kopiert. Sie wurde bei der Subschema-Generierung von UDS erzeugt.

Beispiel für eine Kommandofolge:

```
/SET-FILE-LINK DATABASE,dbname
 /START-PROGRAM $COBOL85
  *COMOPT MODULE=modulbibliothek

  *END quellprogrammdatei
```

Binden eines COBOL-DML-Programms

Das Binden von COBOL-Programmen ist im Kapitel "Erzeugung und Aufruf ablauffähiger Programme" ausführlich beschrieben.

Bei COBOL-DML-Programmen ist jedoch zusätzlich zu beachten, daß je nach Wahl der DBH-Variante (=Database Handler) ein entsprechendes UDS-Connection-Modul mit einzubinden ist (siehe hierzu [14]).

Beispiel eines Binderlaufs:

```
/START-PROGRAM $TSOSLNK
*PROG programmname[,FILENAM=dateiname]
*INCLUDE cobol-dml-programm,modulbibliothek
 *INCLUDE uds-connection-modul,udsmodulbibliothek
 [*RESOLVE ,$.SYSLNK.CRTE]
```

Ablauf eines UDS-Anwenderprogramms

Der Ablauf eines UDS-Anwenderprogramms setzt bei Einsatz des independent DBH eine UDS-Session voraus. Die Verbindung zu dieser Session bzw. zur Datenbank stellt das SET-FILE-LINK-Kommando her.

Ablauf mit linked-in DBH:

```
/SET-FILE-LINK DATABASE,dbname
 /START-PROGRAM dateiname
  [DBH-Parameter]
  PP END
  [Anwenderprogramm-Parameter]
```

Ablauf mit independent DBH:

```
/START-PROGRAM dateiname
  [Anwenderprogramm-Parameter]
```

Übersetzen, Binden und Ablauf von COBOL-SQL-Programmen ist im Handbuch "ESQL-COBOL" [15] beschrieben.

16.3 Beschreibung der Listen

In diesem Abschnitt werden anhand eines Programmbeispiels die Formate folgender Listen kurz erläutert, die COBOL85 im Verlauf einer Übersetzung ausgibt:

– Steueranweisungsliste

– Quellprogrammliste

– Adreß-/Querverweisliste

– Fehlermeldungsliste

In den einzelnen Datensätzen einer Liste sind aus Gründen der Platzersparnis die Leerzeichen nach dem letzten gedruckten Zeichen entfernt.

Überschriftszeile

Jede Seite einer Liste wird von einer Überschriftszeile (siehe unten) eingeleitet, - die unabhängig von der Listenart - folgende Informationen enthält:

(1) Name und Versionsbezeichnung des Compilers

(2) PROGRAM-ID-Name

(3) Listenart

(4) Uhrzeit der Übersetzung

(5) Datum der Übersetzung

(6) Seitennummer

```
     (1)            (2)                          (3)                  (4)      (5)          (6)
COBOL85 V02.3A00    KOPIEREN                     LIBRARY LISTING            14:46:36 2013-09-14  PAGE 0003
```

Steueranweisungsliste

Hier protokolliert COBOL85

(1) die Umgebung des Übersetzungsprozesses,

(2) die ausgewählten Compiler-Optionen (COMOPTs)

(3) die durch Voreinstellung in Kraft befindlichen Compiler-Optionen (COMOPTs) zum Zeitpunkt der Übersetzung und

(4) Informationen für Wartungs- und Diagnosezwecke.

```
COBOL85 V02.3A00  KOPIEREN                          COMOPT LISTING            14:46:36 2013-09-14  PAGE 0001

ENVIRONMENT ———————————————— (1)

                          PROCESSOR                    : 7.500-  S170-30

                          OPERATING SYSTEM             : BS2000 V16.0

                          COMPILER                     : COBOL85  V02.3A00

                          TASK-SEQUENCE-NO             : 2A5K

                          USER-ID                      : H2610491

                          Copyright (C) FUJITSU TECHNOLOGY SOLUTIONS   2009
                                        All Rights Reserved

OPTIONS IN EFFECT ————————————— (2)

                          SYMTEST                      = ALL
                          SYSLIST                      = (OPTIONS,DIAG,MAP,SOURCE,XREF)
                          FLAG-OBSOLETE                = YES
                          LINES-PER-PAGE               = 070
                          FLAG-NONSTANDARD             = YES
                          MERGE-DIAGNOSTICS            = YES
                          GENERATE-SHARED-CODE         = YES

OPTIONS BY DEFAULT ———————————— (3)

                          MODULE                       = *OMF
                          EXPAND-COPY                  = YES
                          LINE-LENGTH                  = 132
                          EXPAND-SUBSCHEMA             = YES
                          MINIMAL-SEVERITY             = I
                          REPLACE-PSEUDOTEXT           = YES
                          RESET-PERFORM-EXITS          = YES
                          CONTINUE-AFTER-MESSAGE       = YES


FOR CUSTOMER SERVICE ————————————— (4)

REV# = A
REV# = B
```

Quellprogrammliste

Jede Zeile einer Quellprogrammliste ist in die folgenden Bereiche unterteilt:

(1) Anzeigenfeld

Spalte 1 informiert über Fehler innerhalb der vom Benutzer vergebenen Numerierung der Eingabesätze (Anzeige S) und über Verstöße gegen die maximale Zeilenlänge von 80 Zeichen (Anzeige T). Außerdem werden in ihm Sätze gekennzeichnet, die aus einer COPY-Bibliothek kopiert wurden (Anzeige C), die durch ein REPLACING bzw. REPLACE vereinbart wurden (Anzeige R) oder die zur SUB-SCHEMA-SECTION gehören (Anzeige D).
In Spalte 3 wird bei expandierten COPY-Elementen die Schachtelungstiefe angezeigt.

(2) Folgenummernfeld

Enthält eine von COBOL85 vergebene, maximal 5-stellige Nummer, die zur Kennzeichnung des eingegebenen Quellprogrammsatzes dient. Diese Nummer dient zur eindeutigen Identifizierung der Quellcodezeilen. Sie findet sich in allen von COBOL85 erzeugten Listen als Querverweisnummer wieder und wird zur Verknüpfung mit etwaigen Fehlermeldungen verwendet. Der maximale Wert beträgt 65535. Überschreitet ein Quellprogramm diese Zahl, wird wieder von 0 an numeriert.

(3) Zu Beginn jeder Seite einer Quellprogrammliste wird nach der Überschrift eine Zeile erzeugt, die Spaltenmarkierungen (V) enthält. Diese Markierungen entsprechen dem COBOL-Referenzformat und erleichtern es dem Benutzer, eine Verletzung des von COBOL geforderten Spaltenformats zu erkennen.

(4) Vom Programmierer nutzbarer Bereich zur Markierung von Quellprogrammzeilen

(5) Quellprogrammbereich

Enthält den vom Benutzer eingegebenen Satz. Dabei ist zu beachten, daß nur abdruckbare Zeichen dargestellt werden.

```
COBOL85   V02.3A00  KOPIEREN                          SOURCE LISTING             14:46:36 2013-09-14  PAGE 0002


(1) (2)  (3)(4)  (5)
          V     VV   V                                                                V
    00001        IDENTIFICATION DIVISION.
    00002        PROGRAM-ID. KOPIEREN.
    00003        ENVIRONMENT DIVISION.
    00004        INPUT-OUTPUT SECTION.
    00005        FILE-CONTROL.
    00006            SELECT SAM-DATEI ASSIGN TO "EINGABE"
    00007                   ORGANIZATION IS SEQUENTIAL
    00008                   FILE STATUS IS SAM-DATEI-ZUSTAND.
    00009            SELECT ISAM-DATEI ASSIGN TO "AUSGABE"
    00010                   ORGANIZATION IS INDEXED
    00011                   RECORD KEY ISAM-SCHLUESSEL
    00012                   FILE STATUS IS IDATSTA
    00013                   ACCESS IS SEQUENTIAL.
    00014        DATA DIVISION.
    00015        FILE SECTION.
    00016        COPY SAM-DATEI.
C 1 00017        FD  SAM-DATEI RECORD IS VARYING IN SIZE FROM 1 TO 255
C 1 00018                              DEPENDING ON SAM-SATZ-LAENGE.
C 1 00019        01  SAM-SATZ.
C 1 00020          05 ZEICHEN       PIC X OCCURS 1 TO 255
C 1 00021                                DEPENDING ON I-LAENGE.
    00022        COPY ISAM-DATEI REPLACING ZEICHEN BY ====.
C 1 00023        FD  ISAM-DATEI RECORD IS VARYING IN SIZE FROM 9 TO 263
C 1 00024                               DEPENDING ON ISAM-SATZ-LAENGE.
C 1 00025        01  ISAM-SATZ.
C 1 00026          05 ISAM-SCHLUESSEL     PIC 9(8).
C 1 00027          05 SATZ-INHALT.
C 1 00028           10 PIC X OCCURS 1 TO 255 DEPENDING ON I-LAENGE.
    00029        WORKING-STORAGE SECTION.
    00030        01  SAM-DATEI-ZUSTAND  PIC XX.
    00031          88 DATEI-ENDE        VALUE "10"
    00032        01  IDATSTA      IS EXTERNAL PIC XX.
   >>>>>  31018 >>>>> 1    PERIOD MISSING AFTER DATA DESCRIPTION ENTRY. PERIOD ASSUMED.
    00033        01  I-LAENGE                 PIC 9(3) BINARY.
    00034        01  ISAM-SATZ-LAENGE         PIC 9(3) BINARY.
    00035        01  SAM-SATZ-LAENGE          PIC 9(3) BINARY.
    00036        PROCEDURE DIVISION.
    00037        ABLAUF SECTION.
    00038        ABLAUF-001.
    00039            OPEN INPUT SAM-DATEI
    00040                 OUTPUT ISAM-DATEI
    00041            PERFORM WITH TEST AFTER
    00042                    VARYING ISAM-SCHLUESSEL FROM 100 BY 100
    00043                    UNTIL DATEI-ENDE
    00044                       OR ISAM-SCHLUESSEL NOT LESS 99999900
    00045              MOVE 255 TO I-LAENGE
    00046              READ SAM-DATEI INTO SATZ-INHALT
    00047                   AT END EXIT TO TEST OF PERFORM
   >>>>>  71113 >>>>> F    COLUMNNUMBER '26': 'EXIT-STATEMENT' IS "NONCONFORMING NONSTANDARD".
    00048              END-READ
    00049              ADD 8 TO SAM-SATZ-LAENGE GIVING ISAM-SATZ-LAENGE
    00050              WRITE ISAM-SATZ
    00051                    INVALID KEY CALL "EAFEHLER"
    00052              END-WRITE
    00053            END-PERFORM
    00054            CLOSE SAM-DATEI ISAM-DATEI
   >>>>>  71168 >>>>> 1    PERIOD MISSING BEFORE PARAGRAPH OR SECTION. PERIOD ASSUMED.
    00055        ABLAUF-999.
    00056            STOP RUN.
```

Im Listenbeispiel sind die Übersetzungsmeldungen "eingemischt", wie in 3.3.6 (LISTING-Option) beschrieben.

Als zweiter Teil der Quellprogrammliste wird eine Bibliotheksliste ausgegeben. Ihr sind die Quellen zu entnehmen, aus denen das in dieser Übersetzung bearbeitete COBOL-Programm entstand. Für jede COPY-Anweisung wird eine Zeile angelegt, die folgende Informationen enthält:

(6) Folgenummer der Quellprogrammzeile, in der die COPY-Anweisung auftritt

(7) Linkname aus der COPY-Anweisung

(8) Dateiname, unter dem die Bibliothek im BS2000-Dateikatalog eingetragen ist.

(9) Bibliothekstyp

(10) Elementname

(11) Datum und

(12) Versionsnummer, mit der das Bibliothekselement in der Bibliothek eingetragen ist. Datum und Versionsnummer sind nicht immer vorhanden.

```
COBOL85    V02.3A00  KOPIEREN                    LIBRARY LISTING          14:46:36 2013-09-14 PAGE 0003


  (6)     (7)       (8)                                                (9)       (10)         (11)       (12)

 SOURCE LIBRARY- FILE-NAME                                            (LIB-) ELEMENT-NAME   USER       VERSION
 SEQ-NO NAME                                                          ORG                   DATE

 SOURCE          :X:$H2610491.KOPIEREN                                ISAM
 00016  COBLIB   :X:$H2610491.COBOL-BIBLIOTHEK                       PLAM   SAM-DATEI      2013-09-14  -
 00022  COBLIB   :X:$F2610491.COBOL-BIBLIOTHEK                       PLAM   ISAM-DATEI     2013-09-14  -
```

Die Formatsteueranweisungen TITLE, EJECT, SKIP

Der COBOL85-Compiler unterstützt die Formatsteueranweisungen TITLE, EJECT und SKIP. Mit diesen Anweisungen im Quellprogramm kann das Aussehen der Quellprogrammliste beeinflußt werden.

Für alle Formatsteueranweisungen gilt:

– Sie dürfen nicht mit einem Punkt abgeschlossen werden.

– Sie müssen allein im B-Bereich einer Zeile stehen.

– Sie sind unwirksam, wenn sie in der IDENTIFICATION DIVISION stehen (da dort jeder Text im B-Bereich als Kommentar behandelt wird).

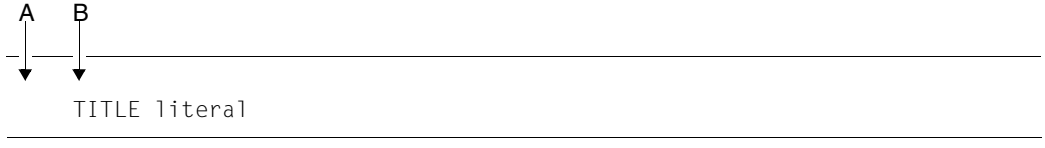
– Sie erscheinen selbst nicht in der Quellprogrammliste.

TITLE-Anweisung

Funktion

Die Anweisung bewirkt, daß nachfolgend in den Kopfzeilen der Quellprogrammliste nicht der Standardtitel (SOURCE LISTING) erscheint, sondern der in der Anweisung angegebene. Zusätzlich wird ein Seitenvorschub erzeugt, wenn nicht ohnehin eine neue Seite beginnt.

Format



Regel

literal muß ein maximal 53 Zeichen langes nichtnumerisches Literal sein.

EJECT-Anweisung

Funktion

Die Anweisung bewirkt, daß der nachfolgende Text der Quellprogrammliste auf der nächsten Seite beginnt. Die Anweisung wirkt nicht, wenn ohnehin eine neue Seite beginnt.

Format


```
    EJECT
```

SKIP-Anweisung

Funktion

Die SKIP-Anweisung dient dazu, den nachfolgenden Text der Quellprogrammliste um bis zu drei Zeilen vorzuschieben. Die Anweisung wirkt nicht, wenn die Leerzeilen als erstes auf einer neuen Seite gedruckt würden.

Format


```
    {SKIP1}
    {SKIP2}
    {SKIP3}
```

Beispiel: Formatsteueranweisungen

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BSP.
DATA DIVISION.
    TITLE "WORKING-STORAGE SECTION"  ------------------------------------ (1)
WORKING-STORAGE SECTION.
01 ALPHA1 PIC 99 VALUE 1.
01 BETA1  PIC 99 VALUE 2.
01 GAMMA1 PIC 99.
    TITLE "PROCEDURE DIVISION"  ----------------------------------------- (2)
PROCEDURE DIVISION.
    EJECT  -------------------------------------------------------------- (3)
ANFANG SECTION.
MULT.
    MULTIPLY ALPHA1 BY BETA1 GIVING GAMMA1.
    MULTIPLY BETA1 BY GAMMA1 GIVING ALPHA1.
    MULTIPLY GAMMA1 BY ALPHA1 GIVING BETA1.
    SKIP3  -------------------------------------------------------------- (4)
ENDE SECTION.
STOPP.
    STOP RUN.
```

Wirkung:

(1) In der Kopfzeile der nächsten Seite der Quellprogrammliste steht "WORKING-STORAGE SECTION"

(2) In der Kopfzeile der nächsten Seite(n) der Quellprogrammliste steht "PROCEDURE DIVISION".

(3) Der nachfolgende Text (ANFANG SECTION...) beginnt auf der nächsten Seite.

(4) Vor dem nachfolgenden Text (ENDE SECTION) stehen drei Leerzeilen.

Fehlermeldungsliste

Die von COBOL85 erzeugte Fehlermeldungsliste gibt Aufschluß über alle während der Übersetzung erkannten Syntax- und Semantikfehler.

Nach der Überschriftszeile unterteilt eine Teilüberschriftszeile die nachfolgenden Fehlermeldungszeilen in folgende Bereiche:

(1)	SOURCE SEQ NO	gibt die Folgenummer der Quellprogrammzeile an, in der der Fehler auftrat.
(2)	MSG INDEX	gibt die Fehlermeldungskennzeichnung an.
(3)	SEVERITY CODE	gibt die Fehlerklasse an (siehe Tabelle 14-1).
(4)	ERROR MESSAGE	enthält den erklärenden Text und gegebenenfalls die von COBOL85 durchgeführte Korrektur oder einen von COBOL85 angenommenen Standardwert.

Am Ende der Fehlermeldungsliste wird eine Abschlußinformation über Gesamtanzahl aller aufgetretenen Fehler sowie Gesamtanzahl der Fehler in den verschiedenen Fehlerklassen ausgedruckt.

```
COBOL85 V02.3A00   KOPIEREN                          DIAGNOSTIC LISTING          14:46:36 2013-09-14  PAGE 0004


   (1)     (2)     (3)         (4)

   SOURCE  MSG   SEVERITY
   SEQ-NO  INDEX     CODE   ERROR MESSAGE

    00032  31018        1    NACH EINER DATENSATZERKLAERUNG FEHLT DER ABSCHLIESSENDE PUNKT.
                             EIN PUNKT WURDE ANGENOMMEN.
    00047  71113        F    SPALTENNUMMER '26': 'EXIT-STATEMENT' IST "NONCONFORMING NONSTANDARD".
    00054  71168        1    VOR EINEM PARAGRAPHEN ODER EINER SECTION FEHLT EIN PUNKT.
                             PUNKT WIRD ANGENOMMEN.


   TOTAL 00003 STATEMENTS IN THIS DIAGNOSTIC LISTING.
         00001 IN SEVERITY CODE F
         00002 IN SEVERITY CODE 1
```

Adreßliste

(1) Angabe des Programmteils, des Kapitels und des Programmnamens

(2) Dateiname, Dateifolgenummer und Adresse des Dateisteuerblocks aller im Programm verwendeten Dateien

(3) SOURCE SEQ-NO
Folgenummer der Quellprogrammzeile, in der die Definition auftritt

(4) MODULE REL ADDR
Relative Anfangsposition einer Datendefinition innerhalb des Moduls

(5) GROUP REL ADDR
Relative Anfangsadresse einer Datendefinition innerhalb einer 01-Stufe (sedezimal).

(6) POSITION IN GROUP DEC
Nummer des ersten Bytes einer Datendefinition innerhalb einer 01-Stufe (dezimal, gezählt ab 1).

(7) LEV NO
Stufennummer der Definition. Ein "G" vor der Stufennummer kennzeichnet ein Datum als "global".

(8) Angabe des vom Benutzer vergebenen Datennamens

(9) LENGTH IN BYTES
Länge des Bereiches, dem der Datenname zugeordnet wurde, in dezimaler (DEC) und in sedezimaler (HEX) Darstellung

(10) FORMAT
Datenklasse in symbolischer Form

(11) REFERENCED BY STATEMENTS
Auflistung aller Quellprogrammzeilennummern in aufsteigender Reihenfolge, in denen Anweisungen stehen, die auf die Datendefinition Bezug nehmen.
Treten mehr Querverweise auf, als in die Zeile passen, wird eine Fortsetzungszeile gebildet (siehe COMOPT LINE-LENGTH, S.75).

(12) LVL
Schachtelungstiefe des Programms, beginnend bei 000 für das äußerste Programm

(13) PROGRAM NAME / SECTION NAME / PARAGRAPH NAME
Angabe des Programmnamens und der in diesem Programm vorhandenen Kapitel- und Paragraphennamen

```
COBOL85  V02.3A00  KOPIEREN                    LOCATOR MAP LISTING              14:46:36 2013-09-14  PAGE 0005


                                        DATA DIVISION            KOPIEREN ———————————————————— (1)
                                        FILE SECTION

                                   FILE NAME               SAM-DATEI ———————————————————— (2)
                                   FILE SERIAL NO.         01
                                   ADDR LHE FCB            000458

 (3)   (4)       (5)      (6)     (7)   (8)                        (9)                 (10)       (11)

       MODULE    GROUP  POSITION                                  LENGTH
SOURCE REL       REL    IN GROUP  LEV                             IN BYTES                      REFERENCED
SEQ NO ADDR      ADDR   DEC        NO                             DEC        HEX       FORMAT   BY STATEMENTS

 00017                              FD SAM-DATEI                                                 00039 00046 ...
 00019 000000B48 000000 00000001    01 SAM-SATZ                    0000000255 000000FF
 00020           000000 00000001    05 ZEICHEN                     0000000001 00000001  CHAR


                                   FILE NAME               ISAM-DATEI
                                   FILE SERIAL NO.         02
                                   ADDR LHE FCB            000830

       MODULE   GROUP  POSITION                                  LENGTH
SOURCE REL      REL    IN GROUP LEV                              IN BYTES                      REFERENCED
SEQ NO ADDR     ADDR   DEC       NO                              DEC        HEX       FORMAT   BY STATEMENTS

 00023                             FD ISAM-DATEI                                                00039 00054
 00025 00000D70 000000 00000001    01 ISAM-SATZ                   0000000263 00000107          00050
 00026 00000D70 000000 00000001    05 ISAM-SCHLUESSEL             0000000008 00000008 ZONED DEC 00011 00041
 00027 00000D78 000008 00000009    05 SATZ-INHALT                 0000000255 000000FF          00046
 00028 00000D78 000008 00000009    10 FILLER                      0000000001 00000001 CHAR
```

```
COBOL85   V02.3A00  KOPIEREN                   LOCATOR MAP LISTING              14:46:36 2013-09-14  PAGE 0006

                                       DATA DIVISION            KOPIEREN
                                       WORKING-STORAGE SECTION

       MODULE   GROUP  POSITION                                  LENGTH
SOURCE REL      REL    IN GROUP LEV                              IN BYTES                      REFERENCED
SEQ NO ADDR     ADDR   DEC       NO                              DEC        HEX       FORMAT   BY STATEMENTS

 00002                            G77 TALLY                      0000000004 00000004 COMP
 00003                            G77 RETURN-CODE                0000000004 00000004 COMP-5
 00030 00000E78 000000 00000001    01 SAM-DATEI-ZUSTAND           0000000002 00000002 CHAR      00008 00041
 00031                             88 DATEI-ENDE                                               00041
 00032 EXTERNAL 000000 00000001    01 IDATSTA                     0000000002 00000002 CHAR      00012
 00033 00000E80 000000 00000001    01 I-LAENGE                    0000000002 00000002 BINARY    00020 00028
 00034 00000E88 000000 00000001    01 ISAM-SATZ-LAENGE            0000000002 00000002 BINARY    00024 00049
 00035 00000E90 000000 00000001    01 SAM-SATZ-LAENGE             0000000002 00000002 BINARY    00018 00049
```

```
COBOL85   V02.3A00  KOPIEREN                   LOCATOR MAP LISTING              14:46:36 2013-09-14  PAGE 0007

                                       PROCEDURE DIVISION

                (12)    (13)                        (11)

                 LVL PROGRAM NAME
SOURCE   REL          SECTION NAME                 REFERENCED
SEQ-NO   ADDR           PARAGRAPH NAME             BY STATEMENTS

                 000 KOPIEREN
 00037   00000000      ABLAUF
 00038   00000000        ABLAUF-001
 00055   00000286        ABLAUF-999
```

Literatur

Die Handbücher finden Sie im Internet unter *http://manuals.ts.fujitsu.com*. Handbücher, die mit einer Bestellnummer angezeigt werden, können Sie in auch gedruckter Form bestellen.

[1] **COBOL85** (BS2000)
COBOL-Compiler
Sprachbeschreibung

[2] **CRTE** (BS2000)
Common RunTime Environment
Benutzerhandbuch

[3] **BS2000/OSD-BC**
Kommandos
Benutzerhandbuch

[4] **BS2000/OSD-BC**
Einführung in das DVS
Benutzerhandbuch

[5] **SDF** (BS2000/OSD)
Dialogschnittstelle SDF
Benutzerhandbuch

[6] **SORT** (BS2000)
SDF-Format

[7] **JV (BS2000) Jobvariablen**
Beschreibung

[8] **AID** (BS2000)
Advanced Interactive Debugger
Testen von COBOL-Programmen
Benutzerhandbuch

[9] **BLSSERV**
Bindelader-Starter
Benutzerhandbuch

[10] **LMS** (BS2000)
SDF-Format
Benutzerhandbuch

[11] **BS2000/OSD-BC**
Systeminstallation
Benutzerhandbuch

[12] **UDS/SQL** (BS2000)
Entwerfen und Definieren
Benutzerhandbuch

[13] **UDS/SQL** (BS2000)
Aufbauen und Umstrukturieren
Benutzerhandbuch

[14] **UDS/SQL** (BS2000)
Anwendungen programmieren
Benutzerhandbuch

[15] **ESQL-COBOL (BS2000) für UDS/SQL**
Benutzerhandbuch

[16] **ESQL-COBOL (BS2000) für SESAM/SQL**
Benutzerhandbuch

[17] System Interfaces for Applications
SQL für ISO/SQL(BS2000)
Portierbare SQL-Anwendungen für BS2000 und SINIX Sprachbeschreibung

[18] **SQL für SESAM/SQL**
Sprachbeschreibung

[19] **SQL für UDS/SQL**
Sprachbeschreibung

[20] **ESQL**
Portierbare ESQL-Anwendungen für BS2000, SINIX und MS-DOS
Benutzerhandbuch

[21] **EDT** (BS2000/OSD)
Anweisungen
Benutzerhandbuch

[22] **AID** (BS2000)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

[23] **AID** (BS2000/OSD)
Testen auf Maschinencode-Ebene
Benutzerhandbuch

[25] **BINDER** (BS2000/OSD)
Benutzerhandbuch

[26] **UTM** (TRANSDATA, BS2000)
Planen und entwerfen
Benutzerhandbuch

[27] **UTM** (TRANSDATA)
Anwendungen programmieren
Basishandbuch inklusive COBOL

[28] **UTM** (TRANSDATA)
Ergänzung für COBOL
Benutzerhandbuch

[29] **UTM**
(BS2000/OSD)
Anwendungen generieren und administrieren
Benutzerhandbuch

[30] **SDF-P (BS2000/OSD)**
Programmieren in der Kommandosprache
Benutzerhandbuch

[31] **POSIX** (BS2000/OSD)
Kommandos
Benutzerhandbuch

[32] **POSIX** (BS2000/OSD)
Grundlagen für Anwender und Systemverwalter
Benutzerhandbuch

[33] **C/C++** (BS2000/OSD)
POSIX-Kommandos des C- und des C++-Compilers
Benutzerhandbuch

Stichwörter

D

E

S

Z