Deutsch

FUJITSU

COBOL2000 V1.6

COBOL-Compiler Benutzerhandbuch

Ausgabe Juni 2018

Inhaltsverzeichnis

COBOL Compiler. Benutzerhandbuch.

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 9001:2015

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 9001:2015 erfüllt.

Copyright und Handelsmarken

1 Einleitung

COBOL2000 ist der COBOL-Compiler für objektorientiertes Programmieren in BS2000/OSD.

1.1 Zielsetzung und Zielgruppen des Handbuchs

Das vorliegende Benutzerhandbuch beschreibt, wie COBOL-Programme in einer BS2000-Systemumgebung verarbeitet werden.

Das Handbuch wendet sich an Benutzer, die über Kenntnisse der Programmiersprache COBOL sowie des Betriebssystems BS2000 verfügen.

1.2 Konzept des Handbuchs

Dieses Benutzerhandbuch beschreibt, wie COBOL-Programme im Betriebssystem BS2000

- für die Übersetzung bereitgestellt,
- mit dem COBOL2000-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.

Darüber hinaus stellt dieses Benutzerhandbuch ab "Verarbeiten von XML-Dokumenten" dar, wie XML-Dokumente verarbeitet werden.

Ferner beschreibt es in Kapitel „COBOL2000 und POSIX" den Einsatz des COBOL2000-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 COBOL2000-Compilers ist im Handbuch „COBOL2000-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.

Readme-Datei

Funktionelle Änderungen der aktuellen Produktversion und Nachträge zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei.

Readme-Dateien stehen Ihnen online bei dem jeweiligen Produkt zusätzlich zu den Produkthandbüchern unter http://manuals.ts.fujitsu.com zur Verfügung. Alternativ finden Sie Readme-Dateien auch auf der Softbook-DVD.

Informationen unter BS2000/OSD

Wenn für eine Produktversion eine Readme-Datei existiert, finden Sie im BS2000-System die folgende Datei:

`SYSRME.<product>.<version>.<lang>`

Diese Datei enthält eine kurze Information zur Readme-Datei in deutscher oder englischer Sprache (<lang>=D /E). Die Information können Sie am Bildschirm mit dem Kommando `/SHOW-FILE` oder mit einem Editor ansehen.
Das Kommando `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` zeigt, unter welcher Benutzerkennung die Dateien des Produkts abgelegt sind.

Ergänzende Produkt-Informationen

Aktuelle Informationen, Versions-, Hardware-Abhängigkeiten und Hinweise für Installation und Einsatz einer Produktversion enthält die zugehörige Freigabemitteilung. Solche Freigabemitteilungen finden Sie online unter http://manuals.ts.fujitsu.com.

1.3 Die Ausbaustufen des COBOL2000-Systems

Das COBOL2000-System V1.6 wird in zwei Ausbaustufen geliefert:

- COBOL2000 (Vollausbau)
- COBOL2000-BC (Basic Configuration / Grundausbaustufe)

In der BC-Version des COBOL2000 werden folgende Steuerungs- und Sprachkomponenten nicht unterstützt:

- symbolisches Testen mit AID
- Ausgabe einer Liste aller Fehlermeldungen
- COBOL-DML-Sprachelemente für Datenbankanschluss
- Sprachmodul Report-Writer
- Compiler- und Programmablauf im POSIX-Subsystem
- Starterphase

Dieses Benutzerhandbuch referiert grundsätzlich die Vollausbaustufe; die Texte zu den von COBOL2000-BC nicht unterstützten Funktionen enthalten einen entsprechenden Hinweis.

Der COBOL-Compiler wird für COBOL2000 Version 1.6 ohne COBOL-Laufzeitsystem ausgeliefert.
Das COBOL-Laufzeitsystem ist Bestandteil des CRTE (Common RunTime Environment), der gemeinsamen Laufzeitumgebung für COBOL-, C- und C++-Programme. Das in CRTE enthaltene COBOL-Laufzeitsystem unterstützt den Ablauf aller Programme, die von COBOL85-Compilern ab Version 1.0A sowie dem COBOL2000-Compiler ab V1.0A übersetzt wurden.

1.4 Änderungen gegenüber der Vorgängerversion

Verarbeitung von XML-Dokumenten

- Bereitstellen des XML-Generators

1.5 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.
{| a | b |}	Senkrechte Balken innerhalb von geschweiften Klammern, schließen Wahlangaben ein, wobei mindestens eine (a oder b) oder auch mehrere Angaben (a und b) ausgewählt werden können. Jede einzelne Alternative soll aber höchstens einmal verwendet werden.
{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.
[| a | b |]	Senkrechte Balken innerhalb von eckigen Klammern, schließen Wahlangaben ein, wobei die Angaben weggelassen werden können oder auch mehrere Angaben ausgewählt werden können. Jede einzelne Angabe soll aber höchstens einmal verwendet werden.
( )	Runde Klammern müssen angegeben werden.
'BLANK'	Dieses Zeichen deutet an, dass mindestens ein Leerzeichen syntaktisch notwendig ist.
Sonderzeichen	Sonderzeichensind ohne Veränderung zu übernehmen.

Hinweis

Für COBOL-Sprachformate gelten die üblichen COBOL-Konventionen (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]).

1.6 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 bindendes 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 Übersetzungseinheiten.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 Übersetzungsprogramms durch den COBOL2000-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 Not-wendigkeit eines Bindelaufs einzugehen, als Objektprogramm bereits das vom COBOL-Compiler erzeugte Kompilat bezeichnet.

Auftrag (Job), Task, Prozess

Ein Auftrag (Job) ist die Folge von Kommandos, Anweisungen etc., die zwischen den Kommandos SET-LOGON-PARAMETERS und EXIT-JOB (bzw. 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 -SET-LOGON-PARAMETERS-Kommando akzeptiert ist.

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

In der Vergangenheit wurde für die Begriffe „Task“ bzw. „Auftrag“ oft synonym der Begriff „Prozess“ verwendet. Im Handbuch werden die oben erklärten Begriffe verwendet. Die Formulierung „bei Prozessende“ bedeutet also: bei Beendigung eines Programmablaufs. Mit „Taskende“ ist der Zeitpunkt nach dem EXIT-JOB- bzw. LOGOFF-Kommando gemeint. Statt des Begriffs „Prozessschalter“ wird der Begriff „Auftragsschalter“ verwendet.

Übersetzungsgruppe

Compilation Group

Eine Folge von Übersetzungseinheiten, die zusammen übersetzt werden.

Übersetzungseinheit

compilation unit

Eine Quelleinheit, die nicht in anderen Quelleinheiten geschachtelt sein kann (Programm-Prototyp, Progammdefinition, Klassendefinition und Interface-Definition). Dies sind die Elemente einer Übersetzungsgruppe. Sie sind separat übersetzbar.

Quelleinheit

source unit

Eine Anweisungsfolge, die mit einer Identification Division beginnt und mit einem zugehörigen END-Eintrag schließt (kann geschachtelt sein).

2 Von der Übersetzungseinheit zum ablauffähigen Programm

Damit aus einer COBOL-Übersetzungseinheit ein ablauffähiges Programm wird, sind drei Schritte nötig:

1. Bereitstellen der Übersetzungseinheit (siehe Abschnitt „Bereitstellen der Übersetzungseinheit")
2. Übersetzen: Die Übersetzungseinheit 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 Kapitel „Binden, Laden, Starten").

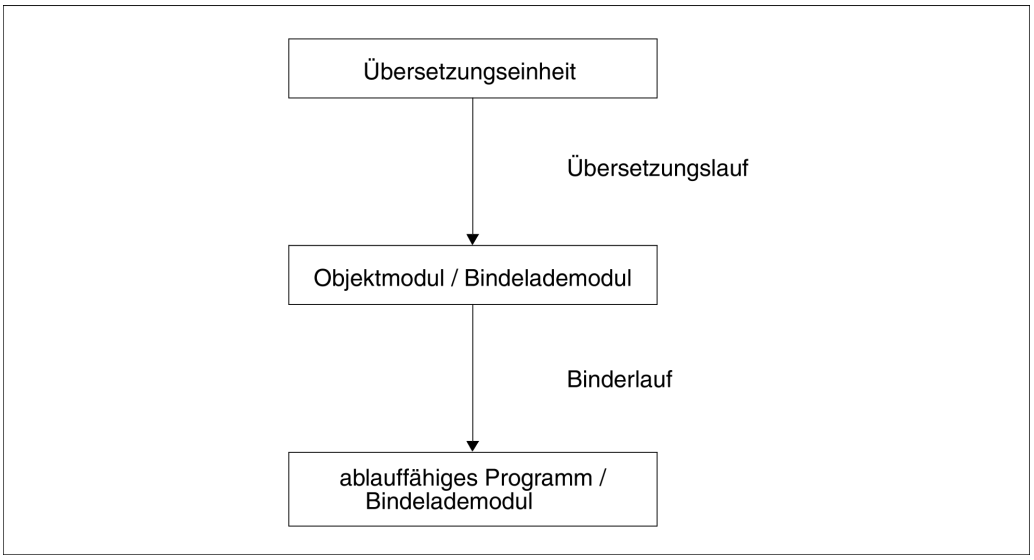


Bild 1: Der Weg zum ablauffähigen Programm

Der Compiler übernimmt während des Übersetzungslaufs drei Funktionen:

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

Durch Steueranweisungen kann der Benutzer

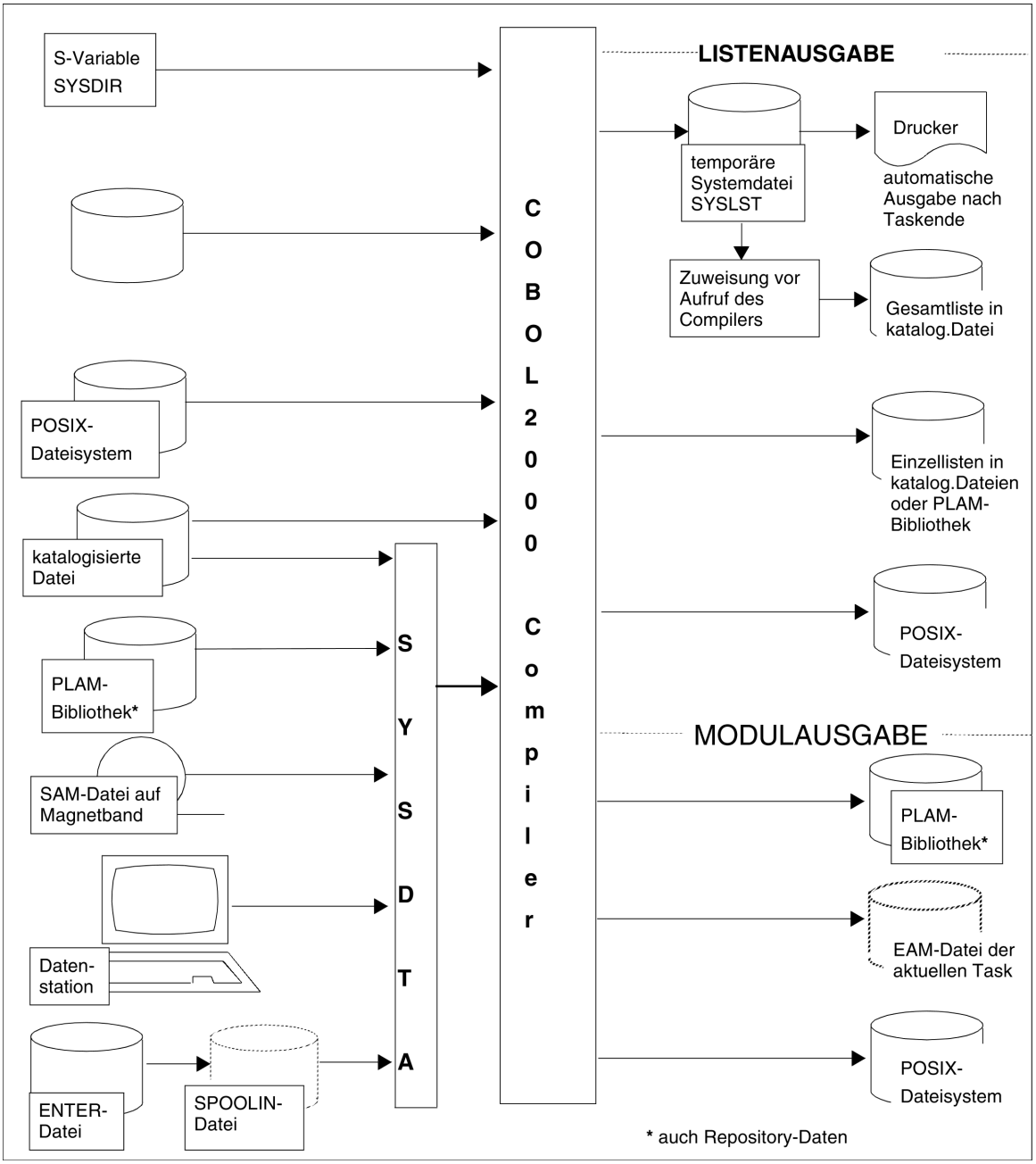
- Funktionen des COBOL2000 auswählen,
- die Betriebsmittel für Ein- und Ausgabe zuweisen,
- Eigenschaften der Module bestimmen,
- Art und Umfang der Listenausgabe festlegen.

Die Steuerungsmöglichkeiten, die COBOL2000 bzw. das Betriebssystem bieten, werden in den Kapiteln „Steuerung des Compilers über SDF" und „Steuerung des Compilers mit COMOPT-Anweisungen" ausführlich beschrieben.

Eine Übersetzungseinheit ist ein COBOL-Quellprogramm, das in **einem** Übersetzungslauf übersetzbar ist. Mit einem einzigen Übersetzungslauf kann aber auch eine Folge von Übersetzungseinheiten, eine so genannte Übersetzungsgruppe, übersetzt werden.

> ℹ Das in den folgenden Kapiteln zu Übersetzungseinheiten Gesagte gilt analog für Übersetzungsgruppen, sofern nicht explizit differenziert wird.

Mögliche Eingabequellen und Ausgabeziele des Compilers



2.1 Bereitstellen der Übersetzungseinheit

Eine COBOL-Übersetzungseinheit muss nach ihrer 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 Übersetzungseinheiten in Dateien oder PLAM-Bibliotheken durch verschiedene Kommandos und Dienstprogramme.

2.1.1 Bereitstellen in katalogisierten Dateien

COBOL2000 kann Übersetzungseinheiten aus SAM- oder ISAM-Dateien verarbeiten, wobei ISAM-Dateien mit KEYPOS=5 und KEYLEN=8 katalogisiert sein müssen. Wie die Übersetzungseinheit in eine solche Datei eingegeben werden kann, hängt davon ab, in welcher Form es zur Verfügung steht:

- Liegt die Übersetzungseinheit bereits auf einem externen Datenträger (z.B. Magnetband) gespeichert vor, kann es mit Hilfe geeigneter
  - BS2000-Kommandos (siehe Handbuch [3]), z.B. des COPY-FILE-Kommandos (für Übersetzungseinheiten auf Magnetbändern),
  - Dienstprogramme, z.B. ARCHIVE für Magnetbänder

  in eine katalogisierte Datei übernommen werden.
- Soll die Übersetzungseinheit neu erfasst werden, lässt sich der Dateiaufbereiter EDT (siehe Handbuch [19]) einsetzen. Er bearbeitet SAM- oder ISAM-Dateien und stellt Funktionen zur Verfügung, die ein formatgerechtes Erstellen und späteres Ändern von COBOL-Übersetzungseinheiten unterstützen. Dazu gehören u.a.
  - die Möglichkeit, einen Tabulator zu setzen: Er erlaubt ein schnelles und zuverlässiges Positionieren auf die Anfangsspalte des Programmtextbereiches und erleichtert so die Einhaltung des Referenzformats für COBOL-Programme (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]).
  - Funktionen für das Einfügen, Löschen, Kopieren, Übertragen und Ändern von Programmierzeilen 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 COBOL2000-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. Für das Einrichten und Verwalten dieser Bibliotheken steht das Dienstprogramm LMS (siehe Handbuch „ LMS (BS2000/OSD)“ [11]) zur Verfügung.

Eine PLAM-Bibliothek kann als Elemente nicht nur Übersetzungseinheiten oder Programmteile (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	Übersetzungseinheiten, COPY-Elemente
R	Objektmodule oder Großmodule
C	ablauffähige Programme
J	Prozeduren
L	Bindelademodule (LLMs)
P	druckaufbereitete Daten (Listen)
X	REPOSITORY-Daten

Tabelle 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 Übersetzungseinheiten aufnehmen

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

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

- Liegt sie in einer katalogisierten Datei oder als Element einer Bibliothek vor, kann sie über das Dienstprogramm LMS in eine PLAM-Bibliothek aufgenommen werden (siehe Beispiel 2-1). Bei der Übernahme einer Übersetzungseinheit aus einer ISAM-Datei mit LMS ist zu beachten, dass mit PAR KEY=YES bzw. SOURCE-ATTRIBUTES=KEEP der ISAM-Schlüssel nicht mitübernommen wird. Eine

Übersetzungseinheit mit ISAM-Schlüssel kann der COBOL2000-Compiler nicht aus einer Bibliothek heraus verarbeiten.

- Soll die Übersetzungseinheit neu erfasst werden, kann sie auch unmittelbar durch den Dateiaufbereiter EDT als Element in eine PLAM-Bibliothek geschrieben werden.

Beispiel 2-1

Übernahme einer Übersetzungseinheit aus einer katalogisierten Datei in eine PLAM-Bibliothek

```
/START-LMS
————————————————————————————————————————————————————————————(1)
% LMS0310 LMS VERSION '03.3A30' STARTED
//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 V03.3A30 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) Die Übersetzungseinheit 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:

- Übersetzungseinheiten (einzelne Übersetzungseinheiten oder Übersetzungsgruppe)
- Programmteile (COPY-Elemente)
- Compiler-Steueranweisungen (COMOPT-Anweisungen oder SDF-Optionen)
- Repository-Daten (Schnittstellenbeschreibungen)

Der Compiler kann Übersetzungseinheiten aus katalogisierten SAM- oder ISAM-Dateien, aus Elementen von PLAM-Bibliotheken und aus POSIX-Dateien verarbeiten. Die Bereitstellung von Übersetzungseinheiten ist im Abschnitt „Bereitstellen der Übersetzungseinheit" und im Kapitel „COBOL2000 und POSIX" beschrieben.

Die Steueranweisungen für die Eingabe sind in den Kapiteln „Steuerung des Compilers über SDF" und „Steuerung des Compilers mit COMOPT-Anweisungen" eingehend beschrieben. Die für beide Steuerungsarten gleiche Zuweisung der Systemdatei SYSDTA ist nachfolgend dargestellt.

2.2.1 Zuweisen der Übersetzungseinheit 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 =] {dateiname | *LIB-ELEM(LIB=library,ELEM=element)}
```

Ausführliche Informationen zum ASSIGN-SYSDTA-Kommando können im Handbuch „BS2000/OSD-BC Kommandos“ [3] nachgelesen werden.

Beispiel 2-2

Einlesen der Übersetzungseinheit aus einer katalogisierten Datei

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

(1) Der Systemdatei SYSDTA wird die katalogisierte Datei QUELL.EINXEINS zugewiesen, in der sich die zu übersetzende Übersetzungseinheit 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 =... spezifiziert wurde.

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

Beispiel 2-3

Einlesen einer Übersetzungseinheit aus einer Bibliothek

```
/ASSIGN-SYSDTA *LIBRARY-ELEMENT(LIB=PLAM.LIB,ELEM=BEISP3)
————————————(1)
Compileraufruf
————————————————————————————————————————————(2)
/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 Compilers mit COMOPT-Anweisungen gebunden. Sie sind in Kapitel „Steuerung des Compilers mit COMOPT-Anweisungen" beschrieben.

2.2.2 Eingabe von Programmteilen

Programmteile (COPY-Elemente) können getrennt von den Übersetzungseinheiten, in denen sie Verwendung finden, in Bibliotheken gespeichert werden. Dies empfiehlt sich vor allem, wenn in verschiedenen Übersetzungseinheiten identische Programmteile vorkommen.
In der Übersetzungseinheit steht stellvertretend für diese Programmteile eine COPY-Anweisung. COPY-Anweisungen dürfen an beliebiger Stelle in der Übersetzungseinheit (außer Kommentarzeilen und nicht-numerischen Literalen) stehen.
Stößt der Compiler beim Übersetzen der Übersetzungseinheit auf eine COPY-Anweisung, holt er aus einer Bibliothek das Element, dessen Name in der COPY-Anweisung angegeben wird. Die COPY-Anweisung wird dann so übersetzt, als wäre das zugehörige Element in der Übersetzungseinheit selbst geschrieben worden.
Das Format der COPY-Anweisung ist im Kapitel zur „Steuerung des Compilers über SDF“ im Handbuch „COBOL2000-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 ADD-FILE-LINK-Kommando zugewiesen und mit den 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 in der Übersetzungseinheit sind folgende Verknüpfungen nötig:

COPY-Anweisung	**ADD-FILE-LINK-Kommando**
`COPY textname` textname bis zu 31 Zeichen langer Elementname	`ADD-FILE-LINK [LINK-NAME=]standard-linkname, [FILE-NAME=]libname` standard-linkname: COBLIB, COBLIB1..COBLIB9 libname: Name der katalogisierten Bibliothek, in der das COPY-Element gespeichert ist
`COPY textname OF bibliothek` bibliothek bis zu 31 Zeichen langer Bibliotheksname	`ADD-FILE-LINK [LINK-NAME=] linkname, [FILE-NAME=] libname` 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 S-Variablen mit dem Standardnamen SYSIOL-COBLIB bzw. SYSIOL-bibliotheksname. Je nach Formulierung der COPY-Anweisung in der Übersetzungseinheit ist die S-Variable folgendermaßen zu gestalten (siehe auch Beispiel 2-6, "Eingabe von Programmteilen"); gilt nicht für BC (Grundausbau):

COPY-Anweisung	**S-Variable**
`COPY textname`	`DECL-VAR SYSIOL-COBLIB,INIT= '*POSIX(pfad)', SCOPE=*TASK`

textname	bis zu 31 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 library`		`DECL-VAR SYSIOL-libname,INIT= '*POSIX(pfad)', SCOPE=*TASK`	
bibliothek	bis zu 31 Zeichen langer Bibliotheksname zur Bildung der S-Variablen mit dem Namen SYSIOL-bibliothek. bibliothek darf keine Kleinbuchstaben enthalten.	libname pfad	die ersten 8 Zeichen von bibliothek 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 BEISPIEL1──────────────────────────────────────────(3)
/ADD-FILE-LINK COBLIB,BIB1────────────────────────────────────────(4)
/ADD-FILE-LINK BIBLIO,BIB2────────────────────────────────────────(5)
```

Compileraufruf

Die Übersetzungseinheit in der Datei BEISPIEL1 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 BEISPIEL1 zugewiesen. Von dort erhält der Compiler die Übersetzungseinheit, in dem zwei COPY-Anweisungen stehen.

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

(5) Das zweite ADD-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.                                                  (1)
      COPY B1.                                                   |
      COPY D1.                                                  (1)
...
```

Zuweisung und Verknüpfung:

```
/ASSIGN-SYSDTA EXAMPLE2 ──────────────────────────────────────── (2)

/ADD-FILE-LINK COBLIB,A                                         (3)
/ADD-FILE-LINK COBLIB1,B                                         |
/ADD-FILE-LINK COBLIB3,D                                        (3)
Compileraufruf ──────────────────────────────────────────────── (4)
```

Die Übersetzungseinheit 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 die Übersetzungseinheit, in der 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.

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 S-Variablen:

```
/DECL-VAR SYSIOL-COBLIB,INIT='*POSIX(/usr/dir1),*POSIX(/usr/dir2)', –
/                           SCOPE=*TASK
–––––––––––––––––––––––––––(2)
/START-COBOL2000-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 „SDF-Menü-Modus"), 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.

2.2.3 Zuweisung an Compilervariablen zur Steuerung der Quelltextmanipulation

Compiler-Direktiven ermöglichen es dem COBOL-Programmierer, die Quelltext-Manipulation zu steuern.

Folgende Compiler-Direktiven stehen zur Verfügung:

- DEFINE-Direktive
- EVALUATE-Direktive
- IF-Direktive

Die Compiler-Direktiven sind ausführlich beschrieben im Handbuch „COBOL2000-Sprachbeschreibung“ [1].

Mit der DEFINE-Direktive kann der Programmierer im Quellprogramm Compilervariablen definieren. Mit Hilfe von S-Variablen kann er diesen Compilervariablen auch vor der Übersetzung Werte zuweisen. Hierfür müssen die Variablen im Programm mit dem Zusatz AS PARAMETER definiert werden. Die Zuordnung der Compilervariablen zur S-Variablen erfolgt über den Namen der Variablen, der wie folgt zu bilden ist:

DEFINE-Direktive	**S-Variable**
`>>DEFINE variable AS PARAMETER`	`DECL-VAR SYSDIR-variable ...,SCOPE=*TASK`

Die S-Variablen sind dabei mit SCOPE=*TASK zu vereinbaren.

Für die Versorgung der Compilervariablen von außen stehen zwei unterschiedliche Typen von S-Variablen zur Verfügung, die mit dem gewünschten TYPE zu deklarieren sind:

- numerische Variablen mit TYPE=*INTEGER
- alphanumerische Variablen mit TYPE=*STRING

Die beiden folgenden Beispiele zeigen die Verwendung von Compilervariablen im BS2000/OSD. Die Verwendung von Compilervariablen beim Compiler-Aufruf unter POSIX ist auf "Übersetzen" beschrieben.

Beispiel 2-7

Übergabe eines numerischen Wertes

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1
...
>>DEFINE VLADIMIR AS PARAMETER.
————————————————————————————————— (1)
...
```

Zuweisung und Verknüpfung:

```
/DECLARE-VARIABLE SYSDIR-VLADIMIR(TYPE=*INTEGER),SCOPE=*TASK
————————————— (2)
/SET-VARIABLE SYSDIR-VLADIMIR=1234
————————————————————————————————— (3)
```

Compiler-Aufruf

(1)

Mit der DEFINE-Direktive wird eine Compilervariable angegeben, deren Inhalt der COBOL-Compiler in einer S-Variablen erwartet.

(2) Mit dem SDF-P-Kommando DECLARE-VARIABLE wird eine S-Variable vereinbart: VLADIMIR ist der Name der numerischen Compilervariablen im Quellprogramm. Die zugehörige S-Variable wird vereinbart als SYSDIR-VLADIMIR mit TYPE=*INTEGER.

(3) Mit dem SDF-P-Kommando SET-VARIABLE wird der S-Variablen SYSDIR-VLADIMIR der numerische Wert 1234 zugewiesen.

Beispiel 2-8

Übergabe eines alphanumerischen Literals

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG2
...
>>DEFINE JERRY AS PARAMETER.
————————————————————————————————————(1)
...
```

Zuweisung und Verknüpfung:

```
/DECLARE-VARIABLE SYSDIR-JERRY (TYPE=*STRING),SCOPE=*TASK
————————————————(2)
/SET-VARIABLE SYSDIR-JERRY='Das ist ein
String'————————————————————————(3)
```

Compiler-Aufruf

(1) Mit der DEFINE-Direktive wird eine Compilervariable angegeben, deren Inhalt der COBOL-Compiler in einer S-Variablen erwartet.

(2) Mit dem SDF-P-Kommando DECLARE-VARIABLE wird eine S-Variable vereinbart: JERRY ist der Name der alphanumerischen Compilervariablen im Quellprogramm. Die zugehörige S-Variable wird vereinbart als SYSDIR-JERRY mit TYPE=*STRING.

(3) Mit dem SDF-P-Kommando SET-VARIABLE wird der S-Variablen SYSDIR-JERRY der alphanumerische Wert „Das ist ein String“ zugewiesen. Die begrenzenden Anführungszeichen sind **nicht** Bestandteil des Literals.

2.3 Ein-/Ausgabe für Repositories

In diesem Kapitel werden folgende Themen behandelt:

- Prinzip des Repository
- Zuweisung eines Repository

2.3.1 Prinzip des Repository

Zur Übersetzung von objektorientierten COBOL-Programmen ist eine externe Bibliothek (logisch eine Bibliothek), REPOSITORY genannt, nötig, die die Beschreibung von Schnittstellen von Programmen, Klassen und Interfaces enthält. Ein Repository muss auch bei nicht objektorientierten Programmen verwendet werden, nämlich dann, wenn bei CALL die Schnittstellen geprüft werden sollen (siehe CALL-Anweisung, Format 3 im Handbuch „COBOL2000-Sprachbeschreibung“ [1]). Diese Beschreibungen werden vom COBOL-Compiler gelesen, um bereits zur Übersetzungszeit zusätzliche Prüfungen durchführen zu können, mit dem Ziel, Fehler beim Ablauf auszuschließen.
Physikalisch ist ein Repository nicht notwendigerweise eine einzige Bibliothek, sondern ggf. eine ganze Hierarchie, vergleichbar den COPY-Bibliotheken.
Repository-Daten sind sowohl Eingabe- als auch Ausgabe-Daten.

2.3.2 Zuweisung eines Repository

Bei einer Übersetzung können zwei Repositories verwendet werden:

- zur Eingabe; dort werden die genutzten Schnittstellen gesucht; hierbei ist eine Hierarchie von Bibliotheken möglich
- zur Ausgabe der Schnittstellenbeschreibung des gerade übersetzten Quelltextes; hierbei ist nur eine einzige Bibliothek möglich.

Die Repository-Daten werden in PLAM-Bibliotheken abgelegt. Der Typ der Elemente ist X (siehe Abschnitt „Bereitstellen in PLAM-Bibliotheken").

Über ADD-FILE-LINK-Kommandos können mehrere Linknamen für Dateien angegeben werden, aus welchen Einträge des REPOSITORY importiert werden sollen.Diese Linknamen sind: REPLIB, REPLIB1,...,REPLIB9. Sie müssen vom Anwender vor dem Start des Compilers im BS2000 zugewiesen werden. Sie werden in der angegebenen Reihenfolge durchsucht, bis eine passende Schnittstellenbeschreibung gefunden worden ist. Wird in diesen Bibliotheken kein Repositoryeintrag gefunden oder ist keine Bibliothek angegeben, so wird in der Bibliothek SYS.PROG.LIB gesucht.

Der über ein ADD-FILE-LINK-Kommando zugewiesene Linkname für die Bibliothek, in die die Ausgabe einer Schnittstelle erfolgen soll ist REPOUT. Dabei kann diese Bibliothek auch eine der Eingabebibliotheken sein. Ist kein Linkname angegeben, so wird auch hier die Bibliothek SYS.PROG.LIB verwendet.Eine Ausgabe findet nur statt, wenn UPDATE-REPOSITORY=YES angegeben ist.

2.4 Ausgaben des Compilers

In diesem Kapitel werden folgende Themen behandelt:

- Ausgabe von Modulen
- Ausgabe von Listen und Meldungen

2.4.1 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 der

Übersetzungseinheit 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 Handbuch [11]).

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

```
/DELETE-SYSTEM-FILE SYSTEM-FILE= *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 „MODULE-OUTPUT-Option" beschrieben.

Bildung von Elementnamen bei der Ausgabe von Modulen in Bibliotheken

Übersetzungseinheit	Modulformat OM	Modulformat LLM
	Standardname abgeleitet aus	
nicht gemeinsam benutzbarer Code[1)] nicht segmentiert	ID-Name [2)] 1..8 [3)]	ID-Name [2)] 1..30 [4)]
segmentiert	PROGRAM-ID-Name 1..6 + Segmentnummer (für jedes Segment)	PROGRAM-ID-Name 1..30 [4)] (Segmentierung ignoriert)
gemeinsam benutzbarer Code [5)]	ID-Name [2)] 1..7@ (Code-Modul) ID-Name [2)] 1..7 (Datenmodul)	ID-Name [2)] 1..30 [3)]

Tabelle 2: Elementnamenbildung bei Modulausgabe

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

2) ID-Name ist PROGRAM-ID-Name, CLASS-ID-Name oder INTERFACE-ID-Name

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

4) 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.
(Nicht für Programmfolgen zulässig).

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

2.4.2 Ausgabe von Listen und Meldungen

Ausgabe von Listen

Der Compiler kann folgende Protokoll-Listen des Übersetzungslaufs erzeugen:

Steueranweisungsliste	OPTION LISTING
Übersetzungseinheitliste	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 Handbuch [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 „Steuerung des Compilers über SDF") oder
- den COMOPT-Anweisungen LISTFILES, LIBFILES oder SYSLIST (siehe Kapitel „Steuerung des Compilers mit COMOPT-Anweisungen").

Falls das POSIX-Subsystem vorhanden ist, können die Listen (außer der Objektliste) ins POSIX-Dateisystem ausgegeben werden. Diese Möglichkeit ist in Abschnitt „LISTING-Option" 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 „Meldungen des COBOL2000-Systems" enthält die kommentierten Texte aller vom Compiler ausgegebenen COB90xx-Meldungen.

2.5 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 COBOL2000-Compiler steuern.
Der COBOL2000-Compiler kann auf verschiedene Arten gesteuert werden:

- durch Optionen im SDF-Syntaxformat
- durch COMOPT-Anweisungen
- durch Compiler-Direktiven

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

Aufrufkommandos	**Steuerungsart**
/START-COBOL2000-COMPILER optionen	SDF-Steuerung, Expert-Modus
/?	SDF-Steuerung, Menü-Modus
/START-COBOL2000-COMPILER?	SDF-Steuerung, Menü-Modus
/START-PROGRAM *name Compilerphase* bzw.*name Starterphase* *)	COMOPT-Steuerung
/START-COBOL2000-COMPILER	keine, Eingabe der Übersetzungseinheit von SYSDTA

Tabelle 3: Aufrufkommando und Steuerungsart

*) gilt nicht für COBOL2000-BC

Die SDF-Steuerung ist in Kapitel „Steuerung des Compilers über SDF", die COMOPT-Steuerung in Kapitel „Steuerung des Compilers mit COMOPT-Anweisungen" ausführlich beschrieben.

Die Steuerung des Compilers im POSIX-Subsystem ist in Kapitel „COBOL2000 und POSIX" beschrieben.

Die Steuerung des Compilers via Compiler-Direktiven ist beschrieben im Kapitel „Steuerung des Compilers mit Compiler-Direktiven" sowie im Handbuch „COBOL2000-Sprachbeschreibung“ [1].

2.6 Beendigung des Compilerlaufs

Das Beendigungsverhalten des COBOL2000-Compilers hängt davon ab,

- welcher Klasse die in der Übersetzungseinheit erkannten Fehler angehören,
- ob der Compiler selbst fehlerfrei abläuft.
  Dieses Verhalten ist vor allem dann von Bedeutung, wenn der COBOL2000-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**	**RückkehrcodeAnzeige in MonitorJobvariablen**	**Auslösen von Spin-Off in Prozeduren** [1)]
keine Fehler	normal	nein	0000	nein
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	nein
Fehlerklasse 3	normal	nein	2005	ja
Compilerfehler	abnormal	ja	3006	

Tabelle 4: Beendigungsverhalten des Compilers

1) Wenn Spin-Off ausgelöst ist, werden alle nachfolgenden Kommandos mit Ausnahme der Kommandos SET-JOB-STEP, EXIT-JOB, LOGOFF, CANCEL-PROCEDURE, END-PROCEDURE und EXIT-PROCEDURE ignoriert. Dabei beendet das Kommando SET-JOB-STEP den Spin-Off und die Verarbeitung wird mit dem nächsten Kommando fortgesetzt.

2.7 Übersetzung von Übersetzungsgruppen

Für die Übersetzung von Übersetzungsgruppen gelten einige Besonderheiten:

Steueranweisungen:

Die vor dem Aufruf des Compilers angegebenen Steueranweisungen gelten für alle Übersetzungseinheiten. Zwischen den Übersetzungseinheiten einer Gruppe 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 jede Übersetzungseinheit ebensoviele Dateien angelegt, wie Listen angefordert wurden.
Bei Verwendung der Standard-Linknamen werden Dateien nach Listenarten angelegt. Die mit OPTLINK verknüpfte Datei enthält eine einzige Optionenliste für alle Übersetzungseinheiten, die mit SRCLINK verknüpfte Datei alle Übersetzungseinheitlisten, die mit ERRLINK verknüpfte Datei alle Fehlerlisten, die mit LOCLINK verknüpfte Datei alle Adress/Querverweislisten.

Listenausgabe in eine PLAM-Bibliothek:

Für jede Übersetzungsgruppe werden ebensoviele Elemente angelegt, wie Listen angefordert wurden (Optionenliste wird nur einmal erzeugt).

Versorgen der Monitor-Jobvariablen:

In der Monitor-Jobvariablen wird stets der Rückkehrcode für diejenige Übersetzungseinheit angezeigt, die den Fehler mit dem höchsten Gewicht enthält.

Compilerabbruch:

Tritt in einer Übersetzungseinheit ein Fehler auf, der zum Abbruch der Übersetzung dieses Programms führt, so wird der gesamte Compilerlauf beendet; d.h. alle nachfolgenden Übersetzungseinheiten werden nicht mehr übersetzt.

Modulausgabe:

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

Repositoryausgabe:

Für jede Übersetzungseinheit wird (sofern verlangt) ein Repositoryeintrag erzeugt.

> ¡ Beim Arbeiten mit Repository (insbesondere, wenn es eine Hierarchie ist) und Repositoryeinträgen, die neu erzeugt werden und in vorhergehenden bzw. nachfolgenden Programmen genutzt werden sollen, muss besondere Sorgfalt darauf verwandt werden, auch wirklich den gewünschten Inhalt zu erhalten.

2.8 Parametrisierte Klassen und Interfaces

Beim Arbeiten mit parametrisierten Klassen bzw. Interfaces sind 3 Schritte zu unterscheiden:

1. **Vorübersetzung** einer parametrisierten Klasse bzw. Interface
2. **Nutzung** einer parametrisierten Klasse bzw. Interface
3. **Expansion** einer parametrisierten Klasse bzw. Interface

Die **Vorübersetzung** einer parametrisierten Klasse bzw. Interface erfolgt ohne Kenntnis der aktuellen Parameter. Ziel dabei ist zum einen die Erkennung von Syntaxfehlern. Zum anderen soll die Schnittstelle und der Quelltext der parametrisierten Klasse bzw. Interface im Repository abgelegt werden (siehe Abschnitt „Ein-/Ausgabe für Repositories"). Der Stand des Quelltextes, der Stand der COPY-Elemente und der Stand der Compiler-Direktiven (siehe Abschnitt „Quelldaten-Eingabe"), die die Übersetzungseinheit der parametrisierten Klasse bzw. Interface gegebenenfalls anspricht, werden zum Zeitpunkt der Vorübersetzung festgehalten und bei den späteren Nutzungen und Expansionen verwendet (d.h. nachträgliche Änderungen daran haben auf die Nutzung und Expansion keine Auswirkung mehr!).

Die **Nutzung** von parametrisierten Klassen bzw. Interfaces erfolgt in der Übersetzungseinheit durch die Verwendung von EXPANDS-Klauseln im REPOSITORY-Paragraf. Somit entstehen neue, konkrete Klassen bzw. Interfaces. Diese bestehen aus den in der Übersetzungseinheit angegebenen aktuellen Parametern und aus den bei der Vorübersetzung festgehaltenen Repository-Daten. Die konkreten Klassen bzw. Interfaces verhalten sich nun wie die nicht-parametrisierten Klassen bzw. Interfaces. Ihre Eigenschaften fließen in die Übersetzung des Nutzers der parametrisierten Klassen bzw. Interfaces ein.

Die **Expansionen** werden automatisch angestoßen im Anschluss an den Nutzer der parametrisierten Klassen bzw. Interface oder bei Übersetzungsgruppen nach der letzten Übersetzungseinheit. Dabei werden alle konkreten Klassen bzw. Interfaces übersetzt, die durch die Nutzung von parametrisierten Klassen bzw. Interfaces entstanden sind. Dazu werden lediglich die bei der Vorübersetzung erzeugten Daten der parametrisierten Klasse bzw. Interface und die aktuellen Parameter (einschließlich deren Repository-Daten) verwendet. Weitere Quelltexte, Bibliothekselemente oder Ähnliches sind für die Expansionen nicht erforderlich. Alle Compileroptionen (siehe Kapitel „Steuerung des Compilers über SDF"), die für die Übersetzungseinheit des Nutzers von parametrisierten Klassen bzw. Interfaces gültig sind, wirken wie bei Übersetzungsgruppen auch für die nachfolgenden Expansionen. Bei Expansionen wird jedoch keine Quellprogrammliste erzeugt. Die Wirkung von >>IMP-Direktiven wird nicht unterdrückt. Im Gegensatz dazu haben Compiler-Direktiven, die für den Nutzer von parametrisierten Klassen bzw. Interfaces gelten, für nachfolgende Expansionen keine Wirkung.

Weitere Details zu parametrisierten Klassen bzw. Interfaces finden Sie im Handbuch „COBOL2000-Sprachbeschreibung“ [1].

Beispiel 2-9

Vorübersetzung einer parametrisierten Klasse

Quelltext

```
CLASS-ID. pkl USING fp.————————————————————————————————(1)
...
REPOSITORY.
CLASS fp.
...
01 obj-fp USAGE OBJECT REFERENCE fp.
01 obj-pkl USAGE OBJECT REFERENCE pkl.
...
```

Zuweisung und Compileraufruf

```
/ADD-FILE-LINK REPOUT,REPOSITORY————————————————————————————(2)
/START-COBOL2000-COMPILER ——————————————————————————————————(3)
/  ... UPDATE-REPOSITORY=*YES ...————————————————————————————(4)
```

(1) Der Name der parametrisierten Klasse ist `pkl`, der Name eines formalen Parameters ist `fp.`

(2) Zur Aufnahme der Repository-Daten wird die Bibliothek REPOSITORY zugewiesen.

(3) Die Vorübersetzung erfolgt durch den COBOL2000-Compiler; der Compiler erkennt selbstständig, ob eine Vorübersetzung durchzuführen ist - eine zusätzliche Steuerung ist dafür nicht erforderlich.

(4) Die Repository-Daten werden als X-Element mit dem Namen PKL$PCL abgelegt (siehe Abschnitt „COMPILER-ACTION-Option").

Beispiel 2-10

Nutzung einer parametrisierten Klasse

Quelltext:

```
PROGRAM-ID. n.
...
REPOSITORY.
CLASS pkl
CLASS exp EXPANDS pkl USING ap————————————————————————————(1)
CLASS ap.
...
01 obj-exp USAGE OBJECT REFERENCE exp.
...
```

Zuweisung und Compileraufruf:

```
/ADD-FILE-LINK REPLIB,REPOSITORY————————————————————————————(2)
/START-COBOL2000-COMPILER ...————————————————————————————————(3)
```

(1) Der Name der Expansion der parametrisierten Klasse ist `exp`, der Name des aktuellen Parameters ist `ap.`

(2) Die Repository-Daten der vorübersetzten parametrisierten Klasse `pkl` sowie der (nicht-parametrisierten) Klasse `ap` werden in der Bibliothek REPOSITORY erwartet.

(3) Im Anschluss an die Übersetzung von n erfolgt automatisch die Übersetzung der konkreten Expansion `exp` der parametrisierten Klasse `pkl`.

ℹ Im Falle von Abhängigkeiten zwischen verschiedenen Expansionen (Beispiel: die Expansion einer parametrisierten Klasse wird als aktueller Parameter für eine andere Expansion verwendet) muss bei der Übersetzung des Nutzers das Eingabe-Repository auch als Ausgabebibliothek zugewiesen und die Option
UPDATE-REPOSIORY=*YES gesetzt werden.

Beispiel 2-11

Expansion einer parametrisierten Klasse

```
temporär erzeugter Quelltext ———————————————————————————— (1)
CLASS-ID. exp USING ap.  ———————————————————————————————— (2)(3)
...
REPOSITORY.
    CLASS ap.  ———————————————————————————————————————————— (3)
...
01 obj-fp USAGE OBJECT REFERENCE ap. ———————————————————— (3)
01 obj-pkl USAGE OBJECT REFERENCE exp. ——————————————————— (2)
...
```

(1) Die Erzeugung und Übersetzung erfolgen automatisch. Der Anwender muss dazu keinerlei zusätzliche Kommandos oder Anweisungen eingeben.

(2) Der Name der parametrisierten Klasse wird an allen Stellen durch den Namen `exp` der konkreten Expansion ersetzt.

(3) Der Name des formalen Parameters wird an allen Stellen durch den Namen `ap` des aktuellen Parameters ersetzt.

> ¡ Der Compiler führt nachfolgende Expansionen nicht zwingend in der Reihenfolge aus, wie sie im Programm geschrieben wurden, sondern so, dass die erforderlichen Daten zu den aktuellen Parametern rechtzeitig vor der Expansion verfügbar sind.

3 Steuerung des Compilers über SDF

Der COBOL2000-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 „Einführung in die Dialogschnittstelle (SDF)“ [5] und „Benutzer-Kommandos (SDF)“ [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-COBOL2000-COMPILER optionen

garantierte Abkürzung: START-COBOL2-COMP optionen
```

Die Übersetzung wird sofort nach Eingabe des Kommandos gestartet.

Falls keine Optionen angegeben werden, liest der Compiler die Übersetzungseinheit von SYSDTA, sofern SYSDTA der Datei bzw. dem Bibliothekselement zugewiesen ist, die die Übersetzungseinheit enthält (siehe Abschnitt „Zuweisen der Übersetzungseinheit mit dem ASSIGN-SYSDTA-Kommando").

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 so weit abgekürzt werden, dass sie innerhalb der jeweiligen SDF-Umgebung eindeutig sind. Unzulässige Abkürzungen und Schreibfehler werden als Syntaxfehler gemeldet und können sofort korrigiert werden.

  > **i** Von der Verwendung von Abkürzungen (insbesondere in Prozeduren) wird jedoch abgeraten, da sich bei zukünftigen Erweiterungen der SDF-Kommandos die möglichen Abkürzungen ändern können.

- 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 zurück schaltet.

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-COBOL2000-COMPILER wird ein Fragezeichen angehängt:

   ```
   /START-COBOL2000-COMPILER? [optionen]

          Wechsel in den Operandenfragebogen
   ```

Mit START-COBOL2000-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-COBOL2000-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 „Einführung in die 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 einerFortsetzungszeile (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 5**: Metazeichen**

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

Kennzeichnung	Bedeutung	Beispiele
GROSSBUCHSTABEN	Großbuchstaben bezeichnen Schlüsselwörter. Schlüsselwörter beginnen mit *.	LISTING = STD SOURCE = *SYSDTA
=	Das Gleichheitszeichen verbindet einen Operandennamen mit dem dazu gehörenden Operandenwert.	LINE-SIZE = 132
< >	Spitze Klammern kennzeichnen Variablen, deren Wertevorrat durch Datentypen und ihre Zusätze beschrieben wird (siehe Tabellen 6 und 7).	... = <integer 1..100>
Unterstreichung	Die Unterstreichung kennzeichnet den Default-Wert 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.	,SHARABLE-CODE = ,ENABLE-INITIAL-STATE=

Tabelle 5: Metazeichen

Tabelle 6: 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 6 beschriebenen Datentypen.

Die Beschreibung der Datentypen gilt für alle Optionen. Deshalb werden bei den entsprechenden Operandenbeschreibungen nur noch Abweichungen von Tabelle6 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.
		$. (Sonderfall)
		System-Standardkennung
		datei
		Datei- oder Jobvariablenname; letztes Zei chen darf kein Bindestrich oder Punkt sein;
		max. 41 Zeichen; muss mindestens ein Zei chen aus A...Z enthalten.
		#datei (Sonderfall)
		@datei (Sonderfall)
		# oder @ als erstes Zeichen kennzeichnet je nach Systemparameter temporäre Datei en 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 angege ben werden.
integer	0...9	

Tabelle 6: Datentypen

Tabelle 7**: 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 7 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 -gen -vers	Schränkt die Angabemöglichkeiten für einen Datentyp ein. Die Angabe einer Dateigeneration oder Dateigenerationsgruppe ist nichterlaubt. 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.

Tabelle 7: Zusätze zu Datentypen

3.3 SDF-Optionen zur Steuerung des Übersetzungslaufs

Name der Option	Zweck
SOURCE	Bestimmen der Eingabequelle der Übersetzungsgruppe
SOURCE-PROPERTIES	Festlegen bestimmter Eigenschaften der Übersetzungsgruppe
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
VERSION	Auswahl des Compilers über seine Versionsnummer

Tabelle 8: Übersicht: Die Optionen zur Steuerung des Compilers

* Option in COBOL2000-BC nicht verfügbar

3.3.1 SOURCE-Option

Die Parameter dieser Option bestimmen, ob die Übersetzungseinheit 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 / <composed-name 1..24>
```

SOURCE = *SYSDTA
Die Übersetzungsgruppe 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 Übersetzungseinheit-Datei 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. Die Datei muss „SYSDTA-fähig“ sein, d.h. ein ASSIGN-SYSDTA-Kommando für diese Datei muss fehlerfrei möglich sein.

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 die Übersetzungsgruppe 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 die Übersetzungsgruppe steht.

VERSION = *HIGHEST-EXISTING / *UPPER-LIMIT / <composed-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 der Übersetzungsgruppe festgelegt werden.

Format

```
SOURCE-PROPERTIES = *STD / *PARAMETERS(...)
    *PARAMETERS(...)
        |  RETURN-CODE = *FROM-COBOL-SUBPROGRAMS / *FROM-ALL-SUBPROGRAMS
        |  ,ENABLE-KEYWORDS=*COBOL85 / *STD(...)
        |       *STD(...)
        |           |  XML-SUPPORT = *YES / *NO
        |  ,STANDARD-DEVIATION=*YES / *NO
        |  ,LINE-SEQUENTIAL=*STD / *SEQUENTIAL
        |  ,XML-NAMES = *KEEP / *UPPER
```

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 des Funktionswertes aus einem Unterprogramm (Register 1) dienen.

ENABLE-KEYWORDS = *COBOL85 / *STD
Bei der Angabe von COBOL85 werden die vom COBOL2000-Compiler zusätzlich gegenüber COBOL85 reservierten Keywords nicht als solche erkannt, sondern können als Datennamen genutzt werden.

XML-SUPPORT = *YES / *NO
Die Angabe YES bewirkt, dass die Schlüsselwörter der neuen Sprachmittel für die XML-Verarbeitung erkannt und diese Sprachmittel übersetzt werden, sowie die Sonderregister zur Verfügung stehen.

Die Angabe NO bewirkt, dass die Schlüsselwörter der neuen Sprachmittel für die XML-Verarbeitung nicht reserviert sind und diese Sprachmittel nicht erkannt werden.

STANDARD-DEVIATION = *YES / *NO
Bei Angabe von YES akzeptiert der Compiler gewisse Abweichungen von den im COBOL-Standard vorgeschriebenen Regeln:

- Den Datenbeschreibungen der Linkage Section (Stufennummer 01 oder 77) **ohne** BASED-Angabe kann ebenfalls mit einer SET-Anweisung die Adresse eines anderen Bereichs oder der Inhalt eines Zeigers zugewiesen werden. Es entfällt dann die Überprüfung, ob jeder benutzte Parameter auch in der USING-Klausel der Procedure Division angegeben wurde.
- Als Empfangsfeld zugelassen sind auch Datenstrukturen, die Zeigerdatenfelder oder universelle Objektreferenzen enthalten.
- Redefiniert werden dürfen auch

- Datenstrukturen, die Zeigerdatenfelder oder universelle Objektreferenzen enthalten, bzw.
- Zeigerdatenfelder oder universelle Objektreferenzen mit Stufennummer 01 oder 77.

- Teilfeldselektiert werden dürfen Datenstrukturen, die Zeigerdatenfelder oder universelle Objektreferenzen enthalten.

> ¡ Bei Angabe von YES ist der Anwender voll verantwortlich, dass Datenstrukturenpassend (auf Wort- bzw. Doppelwortgrenze) ausgerichtet sind.

LINE-SEQUENTIAL = *STD / *SEQUENTIAL
Wird STD angegeben, verarbeitet der Compiler zeilensequenzielle Dateien entsprechend ihren Deklarationen.
Wird SEQUENTIAL angegeben, ignoriert der Compiler das Schlüsselwort LINE in SELECT-Anweisungen und verarbeitet LINE SEQUENTIAL-Dateien als SEQUENTIAL-Dateien.

XML-NAMES = *KEEP / *UPPER
Wird KEEP angegeben, dann überträgt die Anweisung XML GENERATE ein Datenelement ohne Änderungen vom Quelltext zum XML-Element.
Wird UPPER angegeben, werden die Namen von XML-Elementen in Großbuchstaben umgesetzt.

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
```

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.

In den Meldungstexten werden folgende Bezeichnungen verwendet:

„obsolete“	für die veralteten Sprachelemente
„nonconforming nonstandard“	für alle Spracherweiterungen gegenüber ANS85

3.3.4 COMPILER-ACTION-Option

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

Format

```
COMPILER-ACTION = *PRINT-MESSAGE-LIST / *SYNTAX-CHECK / *SEMANTIC-CHECK / *MODULE-
GENERATION(...)
    *MODULE-GENERATION(...)
      |     ,SHAREABLE-CODE = *NO / *YES
      |     ,ENABLE-INITIAL-STATE = *NO / *YES
      |     ,MODULE-FORMAT = *OM / *LLM (...)
      |          LLM (...)
      |               |       ALIGNMENT = *PAGE / *DOUBLE-WORD
      |     ,SUPPRESS-GENERATION = *NO / *AT-SEVERE-ERROR
      |     ,SEGMENTATION = *ELABORATE / *IGNORE
      |     ,UPDATE-REPOSITORY = *NO / *YES
      |     ,CALL-CONVENTION = *COBOL / *COMPATIBLE
      |     ,OPTION-DIRECTIVES = *KEEP / *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 die Übersetzungseinheiten nur auf syntaktische Fehler.

COMPILER-ACTION = *SEMANTIC-CHECK
Der Compiler prüft die Syntax der Übersetzungseinheiten und zusätzlich die Einhaltung der semantischen Regeln. Da kein Modul erzeugt wird, können nur die Übersetzungseinheitliste 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 Abschnitt „Gemeinsam benutzbare COBOL-Programme").

Zur Namensbildung siehe " Ausgabe von Modulen" .
Jede Segmentierung der PROCEDURE DIVISION wird ignoriert.

ENABLE-INITIAL-STATE = *NO / *YES
Bei Angabe von YES legt der Compiler Bereiche für die Initialisierung an. Die Angabe NO bewirkt, dass in Programmen, auf die sich eine CANCEL-Anweisung bezieht, die die INITIAL-Klausel oder INITIALIZE-Anweisungen mit der VALUE-Angabe enthalten, nicht standardkonform ablaufen.

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.

> **i** Bei der Übersetzung von Klassen und Interfaces sollte immer das Format *LLM gewählt werden. Von einander erbende Klassen oder Interfaces müssen alle im gleichen Modulformat vorliegen.

ALIGNMENT = *PAGE / *DOUBLE-WORD
Bei Angabe von PAGE erhalten die CSECTS im generierten Modul das PAGE-Attribut und werden damit auf Seitengrenze ausgerichtet.

Bei Angabe von DOUBLE-WORD werden die CSECTS nur auf Doppelwortgrenze ausgerichtet.

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 und die Expansion von genutzten parametrisierten Klassen bzw. Interfaces unterdrückt werden.

SUPPRESS-GENERATION = *AT-SEVERE-ERROR hat darüber hinaus den Operanden SUPPRESS-GENERATION = *AT-SEVERE-ERROR in der LISTING-Option zur Folge. Damit wird auch die Ausgabe der Objekt-, Adress- und Querverweis-Liste verhindert.

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.

UPDATE-REPOSITORY = *NO / *YES
Bei Angabe von YES legt der Compiler die externe Schnittstelle der Übersetzungseinheiten in das externe Repository ab, das mit dem Linknamen REPOUT zugewiesen ist. Existiert dort bereits eine entsprechende Schnittstelle, wird **nicht** geprüft, ob sich Abweichungen in der Schnittstelle ergeben haben. Die bereits existierende Beschreibung wird durch die neue überschrieben. Existiert kein Link mit dem Namen REPOUT, wird die Bibliothek SYS.PROG.LIB genutzt.

Die Ausgabe erfolgt immer. Eine Unterdrückung über SUPRESS-GENERATION ist nicht möglich. Repository-Daten sind vom Elementtyp X. Zur Unterscheidung erhalten Klassen den Suffix $CLS, Interfaces den Suffix $IFC, parametrisierte Klassen den Suffix $PCL, parametrisierte Interfaces den Suffix $PIF und Programme bzw. ProgrammPrototypen den Suffix $PRO.

CALL-CONVENTION = *COBOL / *COMPATIBLE
Bei Angabe von COBOL wird für die >>CALL-CONVENTION-Direktive der Wert COBOL voreingestellt.

Bei Angabe von COMPATIBLE wird als voreingestellter Wert COMPATIBLE für die >>CALL-CONVENTION-Direktive angenommen.

OPTION-DIRECTIVES = *KEEP / *IGNORE

Bei Angabe von IGNORE werden alle im Quelltext angegebenen >>IMP Direktiven, die Compiler-Optionen betreffen (LISTING-OPTIONS, COMPILER-ACTION und RUNTIME-ERRORS) ignoriert. Damit wird erreicht, dass unabhängig von den im Quelltext angegebenen Direktiven die von außen gesetzten Optionen wirken.

Bei der Expansion von parametrisierten Klassen bzw. Interfaces wird immer OPTION-DIRECTIVES=*KEEP angenommen.

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 / <composed-name 1..
              24>
    |    <composed-name>(...)
    |       | VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING / <composed-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.COBOL.<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.COBOL.<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 im Abschnitt „Ausgabe von Modulen" in Tabelle2 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 3-1

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 = <composed-name 1..24>
Das Element erhält die angegebene Versionsbezeichnung. Soll die Versionsbezeichnung inkrementierbar sein, muss mindestens das letzte Zeichen eine inkrementierbare Ziffer sein (siehe Beispiel 3-1).

ELEMENT = <composed-name 1..32>
Für Bindelademodule (LLMs) kann der Benutzer einen selbstgewählten Elementnamen angeben.

Dieser Operand wird bei der Übersetzung einer Übersetzungsgruppe ignoriert. Statt-dessen werden die Elementnamen der LLMs aus dem jeweiligen PROGRAM-ID-Namen abgeleitet (siehe Abschnitt „Ausgabe von Modulen", Tabelle 2).

VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING / <composed-name 1..24>
Versionsangabe (siehe oben: Versionsangabe für Objektmodule);

Bei der Übersetzung einer Übersetzungsgruppe 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. Pro Übersetzungsgruppe wird nur eine Optionenliste erstellt. Sonstige Listen werden für jede Übersetzungseinheit erzeugt.

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
      |         | ,CROSS-REFERENCE = *NO / *YES
      |         |     *YES(...)
      |         |         | STMT-ADDRESS = *NO / *FIRST
      | ,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 Übersetzungseinheitliste und eine Bibliotheksliste.

COPY-EXPANSION = *NO
Die in die Übersetzungseinheit kopierten COPY-Elemente werden nicht in der Übersetzungseinheitliste abgedruckt. Diese Angabe empfiehlt sich bei häufig vorkommenden COPY-Elementen, um Papier zu sparen.

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

COPY-EXPANSION = *ALL-COPIES
In der Übersetzungseinheitliste werden alle COPY-Elemente abgedruckt, auch diejenigen, die eine SUPPRESS-Angabe enthalten. Jede Zeile eines COPY-Elements ist in Spalte 1 der Liste 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 Übersetzungseinheitliste alle bei der Übersetzung aufgetretenen (Fehler-)Meldungen „eingemischt“. Die Meldungszeile steht dabei jeweils unmittelbar nach der Quellzeile, in der das meldungsauslösende Konstrukt beginnt. Meldungen, die der Compiler keiner bestimmten Quellzeile zuordnen kann, werden nach der letzten Quellzeile ausgegeben.

Der Operand wirkt auch dann, wenn keine Fehlerliste angefordert wurde. Um ein ordnungsgemäßes Einmischen zu gewährleisten, sollte die Übersetzungseinheitliste nicht mehr als 65535 Quellzeilen beinhalten (siehe Abschnitt „Übersetzungseinheitliste").

CROSS-REFERENCE = *YES(...)
Bei Angabe von YES folgen in der Übersetzungseinheitenliste rechts neben den Quellzeilen noch Angaben zu Adresse und Länge von in der Zeile enthaltenen Definitionen, sowie bei Definitionen Querverweise auf die Nutzer einschließlich Nutzungsart und bei den Nutzern Rückverweise auf die Definition.

Der Operand wirkt nicht, wenn die Übersetzungseinheit mehr als 65535 Zeilen umfasst.

Bei Verwendung dieses Operanden empfiehlt es sich, die Zeilenlänge (siehe Operand LAYOUT) zu erhöhen und für das Ausdrucken des Listings dann einen entsprechenden Zeichensatz oder breiteres Papier zu verwenden (siehe Beispiel 3-5).

Bei Zeilen, die in der Übersetzungseinheitenliste nicht aufgelistet werden, entfallen auch die durch den Operand erzeugten zusätzlichen Angaben (siehe COPY-EXPANSION, SUBSCHEMA-EXPANSION und LISTING-Direktive); Verweise aus aufgelisteten Zeilen auf unterdrückte Zeilen bleiben erhalten.

STMT-ADDRESS = *NO / *FIRST
Bei Angabe von FIRST wird in der Übersetzungseinheitenliste rechts neben den Quellzeilen aus der Procedure Division für die erste Anweisung einer Zeile jeweils die Adresse des ersten Maschinenbefehls angegeben, der dafür generiert wurde.

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. Die Angabe YES setzt voraus, dass für ENABLE-KEYWORDS der Wert *COBOL85 angegeben wird.

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 Paragrafennamen.

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 in der Übersetzungseinheit 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.COBOL.*programmname*
Übersetzungseinheitliste/Bibliotheksliste	SRCLST.COBOL.*programmname*
Adressliste/Querverweisliste	LOCLST.COBOL.*programmname*
Fehlerliste	ERRFIL.COBOL.*programmname*

Dateinamen und Dateieigenschaften für diese katalogisierten Dateien sind standardmäßig vorgegeben. Der Benutzer kann aber die Ausgabe in andere katalogisierte Dateien umlenken. Dazu muss er vor dem Aufruf des Compilers die gewünschten Eigenschaften in einem ADD-FILE-LINK-Kommando mit den jeweiligen Dateikettungsnamen (Linknamen) verknüpfen, die der Compiler verwendet:

Liste	Linkname
Steueranweisungsliste	OPTLINK
Übersetzungseinheitliste/Bibliotheksliste	SRCLINK
Adressliste/Querverweisliste	LOCLINK
Fehlerliste	ERRLINK

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

Liste	Name der S-Variablen
Steueranweisungsliste	SYSIOL-OPTLINK
Übersetzungseinheitliste/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.COBOL.*programmname*
Übersetzungseinheitliste/Bibliotheksliste	SRCLST.COBOL.*programmname*
Adressliste/Querverweisliste	LOCLST.COBOL.*programmname*
Fehlerliste	ERRLST.COBOL.*programmname*

programmname wird aus dem PROGRAM-ID-Namen abgeleitet und ggf. auf 16 Zeichen gekürzt. Sollte durch das Abschneiden der Programmname mit einem '-' Zeichen enden, wird das '-' durch das Zeichen '#' ersetzt. 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.

```
/ADD-FILE-LINK ERRLINK,FEHLER
————————————————————————————————————(1)
/START-COBOL2000-COMPILER?
————————————————————————————————————————(2)


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

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

(2) Compileraufruf im Menü-Modus

(3) Die Voreinstellung (Erzeugung von Optionen-, Übersetzungseinheit- 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-COBOL2000-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-, Übersetzungseinheit- 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 Übersetzungseinheit- 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-COBOL2000-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

Beispiel 3-5

Ausgabe eines verdichteten Listings zur Ausnutzung von Druckseiten

Es soll ein verdichtetes Listing erstellt werden, um die Druckseiten so weit wie möglich auszunutzen.

```
LISTING=*PAR(SOURCE=*YES(CROSS-REFERENCE=YES),-
————————————————————————(1)
LAYOUT=*PAR(LINES-PER-PAGE=60,LINE-SIZE=172))                *)
————————————(1)
/PRINT-FILE srclst.cobol.programmname,LOOP=98,CHAR-SET=R01————————————
(2)
```

*) Diese Angaben sind optimiert für eine Seitenbreite von 32 cm und eine Seitenhöhe von 22 cm.

(1) Verwendete Optionen

(2) Kommando zum Ausdrucken der Listing-Datei

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 COBOL2000-BC nicht verfügbar.

Format

```
TEST-SUPPORT = *NONE / *AID(...)

    *AID(...)

        | STMT-REFERENCE = *LINE-NUMBER / *COLUMN-1-TO-6

        | ,PREPARE-FOR-JUMPS = *NO / *YES

        | ,VIRTUAL-NAMES = *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 ID-Paragrafen der Übersetzungseinheit besteht. Beim Testen mit AID kann dieser Name zur Qualifikation der Übersetzungseinheit 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 der Übersetzungseinheit (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 der Übersetzungseinheit (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 „AID“ [8] und Abschnitt „Dialogtesthilfe AID") oder
- Testpunkte gezielt auf Paragrafen 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 Paragrafenüberschriften unmittelbar aufeinander oder auf eine Kapitelüberschrift folgen.
- das AID-Kommando %TRACE jede COBOL-Anweisung einzeln protokollieren soll (siehe Handbuch „AID“ [8]).

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

VIRTUAL-NAMES = *NO / *YES

YES muss angegeben werden, wenn beim Testen mit AID

- FILLER-Elemente von Datenstrukturen angeprochen werden sollen oder
- Einträge im SPECIAL-NAMES paragraph als Elemente der virtuellen Struktur SPECIAL-NAMES ausgegeben werden sollen.

3.3.8 OPTIMIZATION-Option

Mit dieser Option lassen sich die Optimierungsmaßnahmen des Compilers 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 Programm, das ein CALL und/oder CANCEL enthält, sollte mit
RECURSIVE-CALLS=YES übersetzt werden.
Die Option wird für Übersetzungseinheiten, die keine Programme sind, ignoriert. Für Programme mit RECURSIVE-Angabe in der PROGRAM-ID wird sie abgewiesen (bei YES).

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 Abschnitt „LISTING-Option").

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

        | ,EXTRA-ALTERNATE-KEYS = *IGNORE / *STD

        | ,XML-LINE-FEED = *INSERTED / *IGNORED
```

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 Übersetzungseinheit-Zeilennummer der Anweisung enthält, deren Ausführung die Meldung verursacht hat.

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 Abschnitt „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 „COBOL2000 und POSIX" beschrieben.
Dieser Operand ist in COBOL-BC nicht verfügbar.

EXTRA-ALTERNATE-KEYS = *IGNORE / *STD
Voraussetzung für die Verarbeitung einer indizierten Datei mit Sekundärschlüsseln ist standardmäßig (STD) eine identische Beschreibung der Sekundärschlüssel im Programm und im Katalogeintrag der Datei.
Die Angabe von IGNORE bewirkt, dass eine indizierte Datei mit Sekundärschlüsseln auch dann lesend (OPEN INPUT) verarbeitet werden kann, wenn im Katalogeintrag der Datei mehr Schlüssel als im Programm beschrieben sind.

XML-LINE-FEED = *INSERTED / *IGNORED
Die Angabe von INSERTED bewirkt, dass die Satzstruktur einer **Datei**, die ein XML-Dokument enthält, bei der Verarbeitung des Dokuments sichtbar bleibt: Satzwechsel werden in Form von end-of-line-Zeichen an den XML-Parser weitergereicht. Bei Angabe von IGNORED bleibt das Ende eines Dateisatzes für den Parser unsichtbar: Das Dokument erscheint wie ein einziger Satz in der Datei.
Dieser Parameter hat für XML-Dokumente, die im Arbeitsspeicher bereitgestellt werden, keine Wirkung.

> **i** Dieser Parameter bezieht sich nur auf die Satzstruktur einer Datei, nicht jedoch auf end-of-line-Zeichen, die in einem Dateisatz enthalten sind.

3.3.13 VERSION-Option

Mit dieser Option können Sie über die Versionsnummer denjenigen Compiler auswählen, mit dem die Übersetzungsgruppe übersetzt werden soll.

Format

VERSION = *STD / <product-version>

VERSION = *STD
Es wird der zuletzt im System mittels IMON installierte COBOL2000 Compiler aufgerufen.

VERSION = <product-version>
Wählen Sie über die Angabe der Versionsnummer den für die Übersetzung gewünschten Compiler aus, wenn im System mehrere COBOL2000 Compiler mit unterschiedlichen Versionen mittels IMON gleichzeitig installiert sind.
product-version ist dabei in folgendem Format anzugeben: mm.n[a[kk]]

mm	Hauptversion 1..99
n	Nachtragsversion 0..9
a	Modifikationsstatus der Schnittstelle (User Interface) A..Z
kk	Korrektur-Status (Source-/Objektkorrektur) 00..99

Nähere Information zur Installation eines Compilers mittels IMON finden Sie im Imon-Handbuch [34] und im SOLIS Lieferanschreiben.

Bei der Steuerung des Compilers über SDF bestimmt die mit dem als letzten installierten COBOL2000 Compiler aktivierte SDF-Syntaxdatei die verfügbaren Optionen. Um sicherzustellen, dass die neuesten Optionen verfügbar sind, sollten Sie den COBOL2000 Compiler mit der höchsten Versionsnummer zuletzt installieren.

4 Steuerung des Compilers mit COMOPT-Anweisungen

Der COBOL2000-Compiler kann auch wie bisher über COMOPT-Anweisungen gesteuert werden. Er wird zu diesem Zweck mit dem folgenden Kommando aufgerufen:

```
/START-PROGRAM [FROM-FILE =] $.COBOL2000
```

Die Eingabe der Übersetzungseinheit, die Ausgabe der Protokolllisten 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 COBOL2000 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 Übersetzungseinheit-Datei sein - die Optionen stehen dann vor der Übersetzungseinheit - 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 der Übersetzungseinheit 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 der Übersetzungseinheit und liest dort weiter.

Im Batch-Prozess wird bei fehlerhafter Eingabe von COMOPT- oder END-Anweisungen die Übersetzung abgebrochen (Fehlermeldung CBL9005).

Format der COMOPT-Anweisung

```
{COMOPT | COBRUN} operand= {YES | NO | option | (option[,option]...)}
```

- Eingabezeilen 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-Übersetzungseinheiten 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

```
END {filename | libname(elementname)}
```

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 der Übersetzungseinheit beginnen kann.

4.1 Quelldaten-Eingabe bei COMOPT-Steuerung

Eingaben in den Compiler können folgende Quelldaten sein:

- einzelne Übersetzungseinheiten
- Programmteile (COPY-Elemente)
- COMOPT-Anweisungen
- Repository-Daten
- aktuelle Werte für DEFINE-Direktiven

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 Übersetzungseinheiten 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 „Quelldaten-Eingabe" beschrieben. Die Versorgung mit Werten für CompilerDirektiven ist beschrieben im Abschnitt „Zuweisung an Compilervariablen zur Steuerung der Quelltextmanipulation".

4.1.1 Zuweisen der Übersetzungseinheit mit der END-Anweisung

Die Eingabe von Übersetzungseinheiten 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.

> ¡ Falls mit der END-Anweisung ein Bibliothekselement zugewiesen wird, kann der Name der Übersetzungseinheit in den Übersetzungslisten und an der Schnittstelle zu AID-FE nicht korrekt abgebildet werden.
>
> Falls mit der END-Anweisung eine Datei zugewiesen wird, muss diese Datei „SYSDTA-fähig“ sein, d. h. ein ASSIGN-SYSDTA-Kommando muss für diese Datei fehlerfrei möglich sein.

Beispiel 4-1

Zuweisen einer katalogisierten Datei nach Eingabe vonCOMOPT-Anweisungen

```
/START-PROGRAM $.COBOL2000
————————————————————————————————————————(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 die zu übersetzende Übersetzungseinheit 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 $.COBOL2000
————————————————————————————————————————(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.

4.1.2 Zuweisen der Übersetzungseinheit mit ADD-FILE-LINK und COMOPT SOURCE-ELEMENT

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

```
/ADD-FILE-LINK [LINK-NAME=]SRCLIB,[FILE-NAME=]libname
```

Beispiel 4-3

Eingabe aus einer PLAM-Bibliothek

```
/ADD-FILE-LINK SRCLIB,PLAM.LIB
————————————————————————————————————————(1)
/START-PROGRAM $COBOL2000————————————————————————————————————————————————
(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) Die zu übersetzende Übersetzungseinheit steht unter dem Elementnamen BEISP3 in der mit dem ADD-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 MINIMALSEVERITY = 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 eines Unterprogramms dessen Funktionswert (Register 1) im COBOL- Sonderregister RETURN-CODE zur Verfügung steht.

	SDF-Option: SOURCE-PROPERTIES = PARAMETERS(...) RETURN-CODE =
ALIGN-LLM-PAGE={YES/NO} A-L-P	bestimmt, ob CSECTs im generierten Modul auf Seite (YES) oder Doppelwortgrenze (NO) ausgerichtet werden sollen. *Hinweis*: Diese Option ist nur bei LLMs wirksam, nicht jedoch bei OMs. *SDF-Option:* COMPILER-ACTION MODULE-FORMAT=LLM(...) ALIGNMENT=PAGE
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-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=
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. Wirkt nicht für Unterprogramme, die über ENTRY gerufen werden. *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 in der Übersetzungseinheit Satzpaare, die in nicht aufsteigender Reihenfolge gefunden werden, durch eine Meldung der Fehlerklasse 0 (Severity Code 0) in der Fehlerliste gekennzeichnet werden sollen. CHECK-SOURCE-SEQUENCE wirkt nicht bei Free Format. *SDF-Option: --*
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 =
CONCATENATE-XML-LINES={YES/NO} C-X-L	entscheidet, ob die durch die Satzstruktur einer Datei (die ein XML-Dokument enthält) bedingten Zeilenwechsel als end-of-line-Zeichen an den Parser geliefert werden sollen. *SDF-Option:* RUNTIME-OPTIONS=PARAMETERS(...) XML-LINE-FEED=
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 =
DEFAULT-CALL-CONVENTION={COBOL/ COMPATIBLE } DEF-C-C	als voreingestellter Wert wird COMPATIBLE für die CALL-CONVENTION-Direktive angenommen. Bei Angabe von COBOL wird der Wert COBOL für die CALL-CONVENTION-Direktive angenommen. *SDF-Option:* COMPILER-ACTION=MODULE-GENERATION(...) CALL-CONVENTION =*COBOL
ELABORATE-SEGMENTATION={YES/NO}	Bei Angabe von NO werden segmentierungsbezogene Sprachmittel ignoriert (SEGMENT-LIMIT Klausel, Segment-Nummern in Section-Header). Es werden Warnungen ausgegeben. Bei Angabe von YES werden Compiler-Direktiven ignoriert, die während der Übersetzungsphase wirken und innerhalb einer Übersetzungseinheit angegeben sind. YES unterstützt die Segmentierung. Die Übersetzung wird aber mit einer 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-COBOL85-KEYWORDS-ONLY={YES / NO}	Bei Angabe von YES werden nur die für COBOL85 festgelegten Keywords reserviert. Die zusätzlich von COBOL2000 reservierten Schlüsselwörtern können dann als Datennamen genutzt werden. *SDF-Option:* SOURCE-PROPERTIES = PARAMETERS(...) ENABLE-KEYWORDS=

ENABLE-UFS-ACCESS={YES/NO} In COBOL2000-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 =
ENABLE-XML-PROCESSING={YES/NO}	Bei Angabe von NO stehen die neuen Anweisungen zur Verarbeitung von XML-Dokumenten nicht zur Verfügung. Die dafür reservierten Schlüsselwörter können dann als Datennamen genutzt werden. *SDF-Option:* SOURCE-PROPERTIES=PARAMETERS(...) ENABLE-KEYWORDS=STD(...) XML-SUPPORT=
EXPAND-COPY={YES/NO} EXP-COPY	steuert, ob in die Übersetzungseinheit eingefügte COPY-Elemente in der Übersetzungseinheitliste abgedruckt werden. *SDF-Option:* LISTING = PARAMETERS(...) SOURCE = YES(...) COPY-EXPANSION =
EXPAND-SUBSCHEMA={YES/NO} EXP-SUB In COBOL2000-BC nicht verfügbar!	steuert, ob die SUBSCHEMA SECTION der Übersetzungseinheit in der Übersetzungseinheitliste protokolliert wird. *SDF-Option:* LISTING = PARAMETERS(...) SOURCE = YES(...) SUBSCHEMA-EXPANSION =
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 =
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 =
GENERATE-INITIAL-STATE={YES/NO} GEN-INIT-STA	legt fest, ob der Compiler Vorkehrungen treffen soll, dass das Programm erneut in den Initialzustand versetzt werden kann. Alle Programme, die von einer CANCEL-Anweisung betroffen sind, eine INITIAL-Klausel oder eine INITIALIZEAnweisung mit der TO VALUE-Angabe enthalten, sollten für einen standardkonformen Ablauf mit GENERATE-INITIAL-STATE=YES übersetzt werden. *SDF-Option:* COMPILER-ACTION = MODULE-GENERATION(...) ENABLE-INITIAL-STATE =
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 COBOL2000 vergebene Übersetzungseinheit-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 = OM / LLM
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 =
IGNORE-COPY-SUPPRESS={YES/NO} IGN-C-SUP	bestimmt, ob in der Übersetzungseinheit vorhandene COPY-Elemente mit SUPPRESS-Angabe trotzdem in der Übersetzungseinheitliste aufgeführt werden. IGNORE-COPY-SUPPRESS=YES hat darüberhinaus den Operanden EXPAND-COPY=YES zur Folge. *SDF-Option:* LISTING = PARAMETERS(...) SOURCE = YES(...) COPY-EXPANSION =
IGNORE-EXTRA-ALTERNATE-KEYS={YES/NO } IGN-EXT-ALTKEY	bestimmt, ob bei OPEN INPUT für indizierte Dateien Sekundärschlüssel, die nur im Katalogeintrag, aber nicht im Programm beschrieben sind, ohne Fehlermeldung ignoriert werden sollen. *SDF-Option:* RUNTIME-OPTIONS = PARAMETERS(...) EXTRA-ALTERNATE-KEYS =
IGNORE-LINE-SEQUENTIAL={YES/NO}	

	bestimmt, ob der Compiler das Schlüsselwort LINE in SELECT-Anweisungen ignorieren und LINE SEQUENTIAL-Dateien als SEQUENTIAL-Dateien verarbeiten soll. Wird die Standardeinstellung NO verwendet, verarbeitet der Compiler zeilensequentielle Dateien entsprechend ihren Deklarationen. *SDF-Option:* SOURCE-PROPERTIES = PARAMETERS(...) LINE-SEQUENTIAL =
IGNORE-OPTION-DIRECTIVES={YES/NO} IGN-O-DIR	bestimmt, ob >>IMP-Direktiven, die Compiler-Optionen betreffen (LISTING-OPTIONS, COMPILER-ACTION und RUNTIME-ERRORS), ignoriert werden. *SDF-Option:* COMPILER-ACTION = MODULE-GENERATION(...) OPTION-DIRECTIVES =
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.
KEEP-XML-NAMES={YES/NO}	bestimmt, ob die Anweisung XML GENERATE die Namen von Datenelementen vom Quelltext zum XML-Dokument ohne Änderungen (YES) oder in Großbuchstaben umgewandelt (NO) übertragen soll. *SDF-Option:* SOURCE-PROPERTIES = PARAMETERS(...) XML-NAMES =
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 [NO]SOURCE [NO]OBJECT ALL [NO]MAP [NO]XREF NO 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.COBOL.programmname (Steueranweisungsliste) SRCLST.COBOL.programmname (Übersetzungseinheitliste) ERRLST.COBOL.programmname (Fehlerliste) LOCLST.COBOL.programmname (Adress-/Querverweisliste) OBJLST.COBOL.programmname (Objektliste) Zum ggf. nötigen Abschneiden der Namen siehe "LISTING-Option ". Die PLAM-Bibliothek muss mit dem ADD-FILE-LINK Kommando über den Linknamen LIBLINK zugewiesen werden. Wird kein Bibliotheksname zugewiesen, legt der Compiler die angeforderten Listen in der Standard-Bibliothek PLIB.COBOL.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 =
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 [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. Die Angabe von YES setzt voraus, dass ENABLE-COBOL85-KEYWORDS-ONLY ebenfalls auf YES gesetzt ist. *SDF-Option:* LISTING=PARAMETERS(...) DIAGNOSTICS=YES MARK-NEW-KEYWORDS=
MAXIMUM-ERROR-NUMBER=ganzzahl MAX-ERR	legt fest, ab welcher Fehlerzahl (in Abhängigkeit von derMINIMAL-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 Übersetzungseinheitliste „eingemischt“ werden. Um ein ordnungsgemäßes Einmischen zu gewährleisten, sollte die Liste nicht mehr als 65535 Quellzeilen beinhalten (siehe Abschnitt „Übersetzungseinheitliste"). *SDF-Option:* LISTING=PARAMETERS(...) SOURCE=YES(...) INSERT-ERROR-MSG=
MERGE-REFERENCES={YES/NO} M-REF	bewirkt, dass die Übersetzungseinheitenliste erweitert wird mit Angaben zu Adresse und Länge von Definitionen, sowie Querverweisen auf Referenzen bzw. Definitionen.

	SDF-Option: LISTING=PARAMETERS(...) SOURCE=YES(...) CROSS-REFERENCE=
MERGE-STATEMENT-ADDRESS={YES/NO} M-STMT	bewirkt, dass in der Übersetzungseinheitenliste für Anweisungen die Adresse des ersten Maschinenbefehls eingetragen wird, der dafür generiert wurde. (nur wenn die Option MERGE-REFERENCES=YES gesetzt ist) *SDF-Option:* LISTING=PARAMETERS(...) SOURCE=YES(...) CROSS-REFERENCE=YES(...) STMT-ADDRESS=
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 Übersetzungsgruppen 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 Versionsbezeichnung zu geben. version ist eine der folgenden Angaben: *UPPER-LIMIT / *UPPER *HIGHEST-EXISTING / *HIGH *INCREMENT / *INCR <alphanum-name 1..24> *SDF-Option:* MODULE-OUTPUT = *LIBRARY-ELEMENT(...) LIBRARY = <filename>, ELEMENT = <composed-name> VERSION =

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 =
PERMIT-STANDARD-DEVIATION={YES/NO} P-S-D	legt fest, ob • den Datenbeschreibungen der Linkage Section (Stufennummer 01 oder 77) ohne BASED-Angabe ebenfalls mit einer SET-Anweisung die Adresse eines anderen Bereichs oder der Inhalt eines Zeigers zugewiesen werden kann. (Es entfällt dann die Überprüfung, ob jeder Parameter, falls er benutzt wird, auch in der USING-Klausel der Procedure Division angegeben wurde.) • Datenstrukturen, die Zeigerdatenfelder oder universelle Objektreferenzen enthalten, bzw. Zeigerdatenfelder oder universelle Objektreferenzen mit Stufennummer 01 oder 77 als Empfangsfeld zugelassen sind und redefiniert sowie teilfeld-selektiert werden dürfen. *SDF-Option:* SOURCE-PROPERTIES=PARAMETERS(...) STANDARD-DEVIATION=
PRINT-DIAGNOSTIC-MESSAGES={YES/NO} PRI-DIAG In COBOL2000-BC nicht verfügbar!	ermöglicht es, sich eine Liste aller COBOL2000-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. Hexadezimale und nationale Literale werden nicht als einzelne Textwörter erkannt. COPY-Elemente dürfen selbst keine COPY-Anweisungen enthalten. REPLACE-PSEUDOTEXT=NO wird bei Free-Format nicht unterstützt. *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: --*

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. Die Option ist nur wirksam, wenn das Empfangsfeld mit weniger als 19 Dezimalziffern definiert ist. *SDF-Option: --*
SEPARATE-TESTPOINTS={YES/NO} SEP-TESTP In COBOL2000-BC nicht verfügbar!	bestimmt, ob zum Testen mit AID für jeden Paragrafen- 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 COBOL2000-BC nicht verfügbar!	legt fest, ob in der angeforderten Objektliste nur die ESDInformationen 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 EBCDIC-DIN (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 =
SORT-MAP={YES/NO}	gestattet es, sich die Adressliste (LOCATOR MAP) aufsteigend sortiert nach symbolischen Namen aus der Übersetzungseinheit ausgeben zu lassen. Das Protokoll besteht aus Listen für Daten-, Kapitel- und Paragrafennamen. *SDF-Option:* LISTING = PARAMETERS(...) NAME-INFORMATION = YES SORTING-ORDER =
SOURCE-ELEMENT=element SOURCE-ELEM	weist dem Compiler als Übersetzungseinheit ein Element einer PLAM-Bibliothek zu. Vor der Übersetzung muss diese Bibliothek mit dem ADD-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> *SDF-Option:* SOURCE = *LIBRARY-ELEMENT(...) LIBRARY = ,ELEMENT = VERSION =
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 Übersetzungseinheitliste. *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 und die Expansion von genutzten parametrisierten Klassen bzw. Interfaces 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 COBOL2000-BC nicht verfügbar!!	legt die Information fest, die der Compiler für die Dialogtesthilfe AID (siehe Handbuch [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 [NO]DIAG [NO]SOURCE [NO]OBJECT ALL [NO]MAP [NO]XREF NO *SDF-Option:* LISTING = PARAMETERS(...) OUTPUT = SYSLST
TERMINATE-AFTER-SEMANTIC={YES/NO} TERM-A-SEM	ermöglicht es, die Übersetzungsgruppe nur auf syntaktische und semantische Fehler überprüfen zu lassen, ohne dass ein Modul erzeugt wird. Dabei können nur die Übersetzungseinheit- und die Fehlerliste ausgegeben werden. *SDF-Option:* COMPILER-ACTION = SEMANTIC-CHECK
TERMINATE-AFTER-SYNTAX={YES/NO} TERM-A-SYN	ermöglicht es, die Übersetzungsgruppe nur auf syntaktische Fehler überprüfen zulassen, ohne dass ein Modul erzeugt wird. Dabei können nur die Übersetzungseinheitund die Fehlerliste ausgegeben werden. *SDF-Option:* COMPILER-ACTION = SYNTAX-CHECK
TEST-VIRTUAL-NAMES={YES/NO} In COBOL2000-BC nicht verfügbar!	legt fest, ob bei SYMTEST=ALL ob für das Testen mit AID die virtuellen Namen FILLERnn und die Struktur SPECIAL-NAMES generiert werden sollen. *SDF-Option:* TEST-SUPPORT = AID(...) VIRTUAL-NAMES =
TEST-WITH-COLUMN1={YES/NO} TEST-W-C In COBOL2000-BC nicht verfügbar!	legt fest, ob bei SYMTEST=ALL die AID-Source-Referenzen mit Hilfe der Folgenummern der Übersetzungsgruppe (Spalte 1 bis 6) gebildet werden sollen. TEST-WITH-COLUMN1 wird bei Free-Format nicht unterstützt. *SDF-Option:* TEST-SUPPORT = AID(...) STMT-REFERENCE =
UPDATE-REPOSITORY={YES/NO} UPD-R	steuert, ob die Schnittstelle des aktuell übersetzten Quelltextes in das mit dem Linknamen REPOUT zugewiesene externe Repository abgelegt werden soll. Repository-Daten sind vom Elementtyp X. Zur Unterscheidung erhalten Klassen den Suffix $CLS, Interfaces den Suffix $IFC und Programm-Prototypen den Suffix $PRO. *SDF-Option:* COMPILER-ACTION=MODULE-GENERATION UPDATE-REPOSITORY=
USE-APOSTROPHE={YES/NO} USE-AP	steuert die Darstellung der figurativen Konstanten „QUOTE“. Bei ’YES’ hat die figurative Konstante QUOTE das Hochkomma als Wert, bei NO ist es das Anführungszeichen.

5 Steuerung des Compilers mit Compiler- Direktiven

Die >>IMP-Compiler-Direktiven ermöglichen es, einige Übersetzungsoptionen direkt im Quelltext anzugeben.

Im Gegensatz zu Optionen, die für die ganze Übersetzungsgruppe gelten, können Direktiven für jede Übersetzungseinheit separat angegeben werden.

Die durch die SDF-Steuerung bzw. COMOPT-Anweisungen angegebenen Werte definieren die voreingestellten Werte für die im Folgenden beschriebenen Direktiven. Durch die SDF-Option COMPILER-ACTION=*MODULE-GENERATION(OPTION-DIRECTIVES=*IGNORE) bzw. die COMOPT-Anweisung IGNORE-OPTION-DIRECTIVES=YES kann verhindert werden, dass die Außensteuerung durch Direktiven verändert wird.

Die Notation entspricht der COBOL-Notation der Sprachbeschreibung (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]).

5.1 IMP COMPILER-ACTION

Durch diese Direktive können Aktionen des Compilers bei der Modul-Generierung gesteuert werden.

Format

```
 >>IMP COMPILER-ACTION {|GENERATE-INITIAL-STATE | UPDATE-REPOSITORY|} {ON | OFF |
DEFAULT}
```

Syntaxregel

1. Die Direktive darf nur vor einer Übersetzungseinheit angegeben werden.

Allgemeine Regeln

1. Die Direktive wirkt in der Übersetzungsphase.
2. Jeder in der Direktive angegebene Operand bezieht sich auf die gleichnamige CompilerOption (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]).
3. Die Angabe GENERATE-INITIAL-STATE wird abgewiesen, wenn beim Aufruf des Compilers die Compiler-Option RESET-PERFORM-EXITS=NO angegeben wurde.
4. Die Angabe ON bewirkt, dass für die spezifizierte Compiler-Option der Wert YES angenommen wird.
5. Die Angabe OFF bewirkt, dass für die spezifizierte Compiler-Option der Wert NO angenommen wird.
6. Die Angabe DEFAULT bewirkt, dass für die spezifizierte Compiler-Option der Wert, der beim Aufruf des Compilers angegeben war, angenommen wird.

5.2 IMP LISTING-OPTIONS

Durch diese Direktive können die Werte von Compiler-Optionen, die die vom Compiler erzeugten Listen beeinflussen, geändert werden.

Format

```
>>IMP LISTING-OPTIONS {| EXPAND-COPY | EXPAND-SUBSCHEMA | MERGE-DIAGNOSTICS  |
MERGE-REFERENCES | MERGE-STATEMENT-ADDRESS  | SORT-MAP  | SHORTEN-XREF  |}

       {ON |  OFF  |  DEFAULT }
```

Syntaxregel

1. Die Direktive darf nur vor einer Übersetzungseinheit angegeben werden.

Allgemeine Regeln

1. Die Direktive wirkt in der Listen-Erzeugungsphase.
2. Jeder in der Direktive angegebene Operand bezieht sich auf die gleichnamige CompilerOption (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]).
3. Die Angabe ON bewirkt, dass für die spezifizierte Compiler-Option der Wert YES angenommen wird. Die Direktive wirkt jedoch nur, wenn die Liste, die von ihr betroffen wird, auch erzeugt werden soll.
4. Die Angabe OFF bewirkt, dass für die spezifizierte Compiler-Option der Wert NO angenommen wird.
5. Die Angabe DEFAULT bewirkt, dass für die spezifizierte Compiler-Option der Wert, der beim Aufruf des Compilers angegeben war, angenommen wird.
6. Der letzte vor einer Übersetzungseinheit für einen Operanden angegebene Wert wird auch bei der Listen-Erzeugung der vor einer Übersetzungseinheit angegebenen Zeilen herangezogen (z.B. wirkt der letzte Wert für EXPAND-COPY auch für COPY-Anweisungen, die vor dieser Direktive angegeben wurden).

5.3 IMP PRINT-DIRECTIVES

Durch diese Direktive können die Werte von Direktiven in die Quellprogrammliste ausgegeben werden.

Format

```
>>IMP PRINT-DIRECTIVES {ALL | NON-DEFAULT}
```

Allgemeine Regeln

1. Die Direktive wirkt in der Listenerzeugungsphase.
2. Die Direktive darf überall in einer Übersetzungseinheit angegeben werden.
3. Durch die Direktive können die Werte von Direktiven, die in der Übersetzungsphase wirken, sowie der >>IMP LISTING-OPTIONS Direktive, in die Quellprogrammliste ausgegeben werden.
4. Die Angabe ALL bewirkt, dass die Werte aller Direktiven in die Quellprogrammliste ausgegeben werden.
5. Die Angabe NON-DEFAULT bewirkt, dass die Werte aller Direktiven, die vom Defaultwert abweichen, in die Quellprogrammliste ausgegeben werden.
6. Die Ausgabe wird im LISTING direkt im Anschluss an die Direktive erzeugt.
7. Ist für die Zeile, in der die Direktive angegeben ist, die Listenerzeugung ausgeschaltet (>>LISTING OFF, COPY ... SUPPRESS,...), so werden auch die Direktivenwerte nicht aufgelistet.

Beispiel 5-1

Quelltext:

```
        >>IMP PRINT-DIRECTIVES NON-DEFAULT
        >>IMP PRINT-DIRECTIVES ALL
        >>IMP LISTING-OPTIONS MERGE-DIAGNOSTICS
        >>CALL-CONVENTION COBOL
        >>TURN EC-OO-CONFORMANCE EC-OO-NULL CHECKING ON
        >>IMP LISTING-OPTIONS EXPAND-COPY EXPAND-SUBSCHEMA OFF
        >>IMP RUNTIME-ERRORS FUNCTION-DEFAULT-VALUE ON
        >>IMP LISTING-OPTIONS SORT-MAP SHORTEN-XREF ON
        >>IMP PRINT-DIRECTIVES NON-DEFAULT
        >>IMP PRINT-DIRECTIVES ALL
        ...
```

Listenausgabe:
OPTIONS BY DEFAULT

```
                        ...
                        EXPAND-COPY = YES
                        ...
                        EXPAND-SUBSCHEMA = YES
                        ...
                        GENERATE-INITIAL-STATE = YES
                        ...
       V    VV
 00001       >>IMP PRINT-DIRECTIVES NON-DEFAULT
#####  ALL DIRECTIVES VALUES ARE SET TO DEFAULT ####
 00002       >>IMP PRINT-DIRECTIVES ALL
##### >>CALL-CONVENTION
```

```
COMPATIBLE
##### >>FLAG-85 ZERO-LENGTH                                    OFF
##### >>IMP RUNTIME-ERRORS FUNCTION-DEFAULT-VALUE              OFF
##### >>IMP LISTING-OPTIONS EXPAND-COPY                        ON
##### >>IMP LISTING-OPTIONS EXPAND-SUBSCHEMA                   ON
##### >>IMP LISTING-OPTIONS MERGE-DIAGNOSTICS                  OFF
##### >>IMP LISTING-OPTIONS MERGE-REFERENCES                   OFF
##### >>IMP LISTING-OPTIONS MERGE-STATEMENT-ADDRESS            OFF
##### >>IMP LISTING-OPTIONS SORT-MAP                           OFF
##### >>IMP LISTING-OPTIONS SHORTEN-XREF                       OFF
##### >>IMP COMPILER-ACTION GENERATE-INITIAL-STATE             ON
##### >>IMP COMPILER-ACTION UPDATE-REPOSITORY                  OFF
##### >>TURN EC-DATA-CONVERSION                        CHECKING OFF
##### >>TURN EC-OO-CONFORMANCE                         CHECKING OFF
##### >>TURN EC-OO-METHOD                              CHECKING OFF
##### >>TURN EC-OO-NULL                                CHECKING OFF
##### >>TURN EC-OO-RESOURCE                            CHECKING OFF
##### >>TURN EC-OO-UNIVERSAL                           CHECKING OFF
##### >>TURN EC-STORAGE-NOT-ALLOC                      CHECKING OFF
##### >>TURN EC-STORAGE-NOT-AVAIL                      CHECKING OFF
##### >>TURN EC-XML-CODESET-CONVERSION                 CHECKING OFF
 00003       >>IMP LISTING-OPTIONS MERGE-DIAGNOSTICS
 00004       >>CALL-CONVENTION COBOL
 00005       >>TURN EC-OO-CONFORMANCE EC-OO-NULL CHECKING ON
 00006       >>IMP LISTING-OPTIONS EXPAND-COPY EXPAND-SUBSCHEMA OFF
 00007       >>IMP RUNTIME-ERRORS FUNCTION-DEFAULT-VALUE ON
 00008       >>IMP LISTING-OPTIONS SORT-MAP SHORTEN-XREF ON
 00009       >>IMP PRINT-DIRECTIVES NON-DEFAULT
##### >>CALL-CONVENTION                                       COBOL
##### >>FLAG-85 ZERO-LENGTH                                   OFF
##### >>IMP RUNTIME-ERRORS FUNCTION-DEFAULT-VALUE             ON
##### >>IMP LISTING-OPTIONS EXPAND-COPY                       OFF
##### >>IMP LISTING-OPTIONS EXPAND-SUBSCHEMA                  OFF
##### >>IMP LISTING-OPTIONS MERGE-DIAGNOSTICS                 ON
##### >>IMP LISTING-OPTIONS SORT-MAP                          ON
##### >>IMP LISTING-OPTIONS SHORTEN-XREF                      ON
##### >>TURN EC-OO-CONFORMANCE CHECKING                       ON
##### >>TURN EC-OO-NULL CHECKING                              ON
 00010       >>IMP PRINT-DIRECTIVES ALL
##### >>CALL-CONVENTION                                       COBOL
##### >>IMP RUNTIME-ERRORS FUNCTION-DEFAULT-VALUE             ON
##### >>IMP LISTING-OPTIONS EXPAND-COPY                       OFF
##### >>IMP LISTING-OPTIONS EXPAND-SUBSCHEMA                  OFF
##### >>IMP LISTING-OPTIONS MERGE-DIAGNOSTICS                 ON
##### >>IMP LISTING-OPTIONS MERGE-REFERENCES                  OFF
##### >>IMP LISTING-OPTIONS MERGE-STATEMENT-ADDRESS           OFF
##### >>IMP LISTING-OPTIONS SORT-MAP                          ON
##### >>IMP LISTING-OPTIONS SHORTEN-XREF                      ON
##### >>IMP COMPILER-ACTION GENERATE-INITIAL-STATE            ON
##### >>IMP COMPILER-ACTION UPDATE-REPOSITORY                 OFF
##### >>TURN EC-DATA-CONVERSION                       CHECKING OFF
##### >>TURN EC-OO-CONFORMANCE                        CHECKING ON
##### >>TURN EC-OO-METHOD                             CHECKING OFF
##### >>TURN EC-OO-NULL                               CHECKING ON
##### >>TURN EC-OO-RESOURCE                           CHECKING OFF
##### >>TURN EC-OO-UNIVERSAL                          CHECKING OFF
##### >>TURN EC-STORAGE-NOT-ALLOC                     CHECKING OFF
```

```
##### >>TURN EC-STORAGE-NOT-AVAIL                    CHECKING OFF
##### >>TURN EC-XML-CODESET-CONVERSION               CHECKING OFF
```

5.4 IMP RUNTIME-ERRORS

Durch diese Direktive kann die Überprüfung und Behandlung von Laufzeitfehlern gesteuert werden.

Format

```
>>IMP RUNTIME-ERRORS FUNCTION-DEFAULT-VALUE {ON | OFF | DEFAULT}
```

Allgemeine Regeln

1. Die Direktive wirkt in der Übersetzungsphase.
   Ist die Direktive innerhalb einer Anweisung angegeben, so wirkt sie erst für die nächste Klausel bzw. Anweisung.

   > ℹ Für WHEN-Angaben in EVALUATE- bzw. SEARCH-Anweisungen gilt dieselbe RUNTIME-ERRORS-Angabe, wie bei der EVALUATE- bzw. SEARCH-Anweisung selbst.

2. Die Direktive darf überall in einer Übersetzungseinheit angegeben werden.
3. Die Direktive wird abgewiesen, wenn beim Aufruf des Compilers die Compiler-Option CHECK-FUNCTION-ARGUMENTS=YES angegeben wurde.
4. Die Angabe ON bewirkt, dass für die Compiler-Option SET-FUNCTION-ERROR-DEFAULT der Wert YES angenommen wird.
5. Die Angabe OFF bewirkt, dass für die Compiler-Option SET-FUNCTION-ERROR-DEFAULT der Wert NO angenommen wird.
6. Die Angabe DEFAULT bewirkt, dass für die Compiler-Option SET-FUNCTION-ERROR-DEFAULT der Wert, der beim Aufruf des Compilers angegeben war, angenommen wird.

6 Binden, Laden, Starten

Im Verlauf der Übersetzung erzeugt COBOL2000 Objektmodule (OM's) 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 Kapitel „COBOL2000 und POSIX").
- 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 Abschnitt „Gemeinsam benutzbare COBOL-Programme"). Im Datenmodul existiert ein Externverweis auf das zugehörige Codemodul.
- Das COBOL-Programm verwendet Sprachmittel für Objektorientierung.Der Compiler erzeugt Externverweise für alle nicht-parametrisierten Klassen bzw. Interfaces, die im REPOSITORY Paragrafen spezifiziert sind.
- Das COBOL-Programm enthält mit External beschriebene Daten. Der Compiler legt dafür Common-Bereiche an.

Die Verwendung einiger Sprachmittel in Programmen verlangt beim Binden bzw. Laden zusätzliche Module, die nicht Bestandteil des Laufzeitsystems (CRTE) sind, sondern im System verfügbar sein müssen. Diese müssen entweder statisch eingebunden werden (siehe Abschnitt „Statisches Binden mit TSOSLNK" und Abschnitt „Binden mit dem BINDER") oder beim dynamischen Binden über entsprechende Linknamen BLSLIBnn zugewiesen werden (siehe Abschnitt „Dynamisches Binden und Laden mit dem DBL"). Im Einzelnen betrifft das:

- SORT:

  Modul SORT80 mit (u.A.) Entries ILSORT und SORTZM1;
  Dieses Modul ist üblicherweise in der Bibliothek $TASKLIB vorhanden.
- Bibliothekselement als zeilensequenzielle Datei:

  Modul LMSUP1;
  Dieses Modul steht üblicherweise in der Bibliothek $LMSLIB.
- nationale Daten (UTF-16):

  Modul GNLADPT;
  Wo dieses Modul zu finden ist, entnehmen Sie dem Handbuch „XHCS (BS2000/OSD)“ [33].

- XML:
  - Modul GNLADPT;
    Siehe Handbuch „XHCS (BS2000/OSD)“ [33].
  - Modul ITCRXFCA in der ’Parser-Bibliothek’, die Sie bereitstellen müssen; Siehe Kapitel „Verarbeiten von XML-Dokumenten".

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 **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 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.
- 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 „Binden und Laden von Unterprogrammen").
- 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 COBOL2000-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 BinderLader-Starter-Systems verarbeitet bzw. erzeugt werden.

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

Objektprogramm (Lademodul)	nein	nein	ja	ja

*) Nur im Betriebsmodus ADVANCED

Der Bindevorgang im POSIX-Subsystem ist in Kapitel „COBOL2000 und POSIX" erläutert.

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

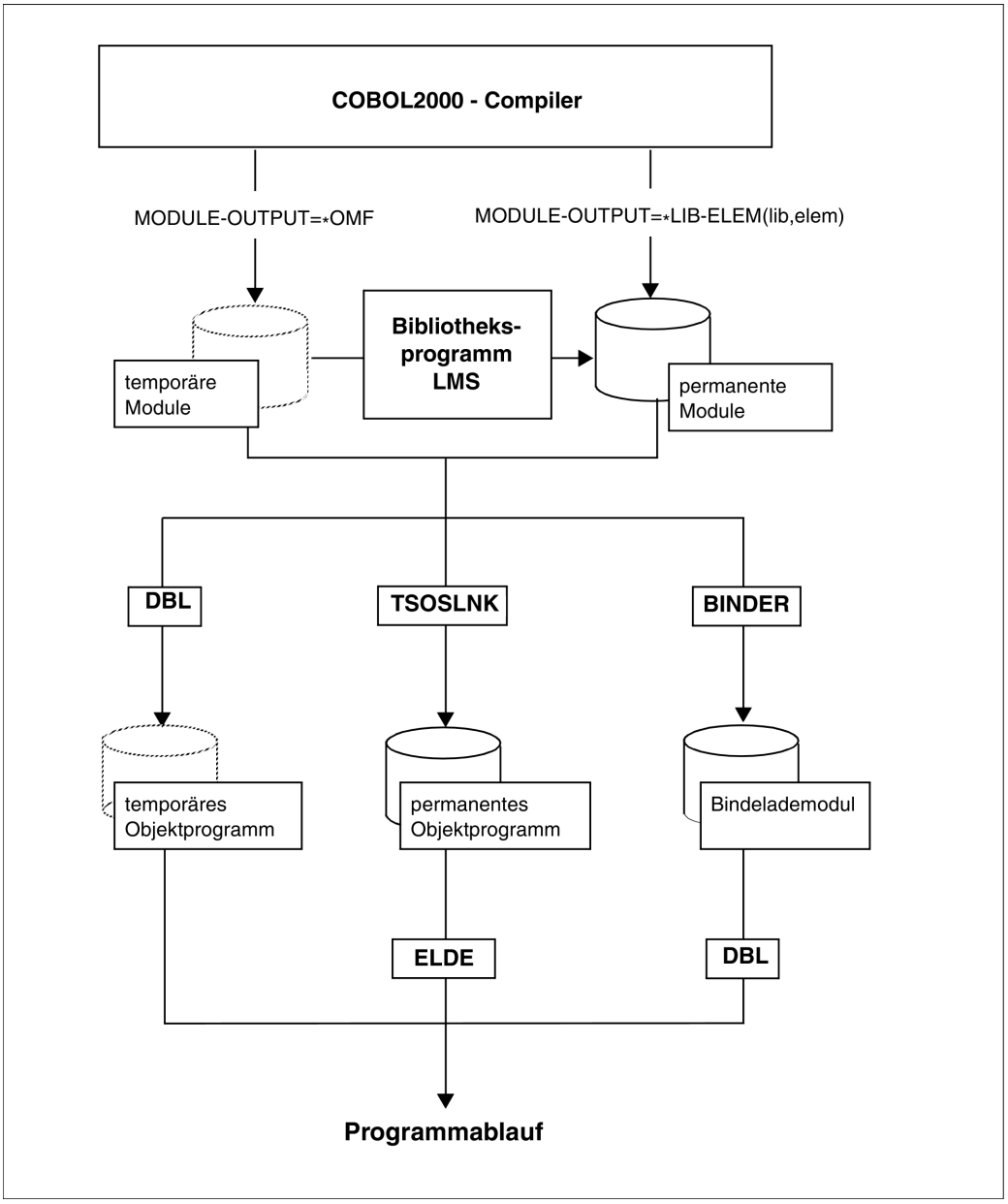


Bild 2: Erzeugung und Aufruf permanent und temporär ablauffähiger COBOL-Programme im BS2000

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

Die Steueranweisungen für TSOSLNK und deren Operanden sind ausführlich im Handbuch „TSOSLNK“ [9] beschrieben; die Zusammenstellung auf der folgenden Seite gibt nur einen Überblick über die wichtigsten Angaben.

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 der Übersetzungseinheit zu verwenden. Voraussetzung dafür ist, dass COBOL2000 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 Handbuch „AID“ [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.e insprungstelle 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.

Tabelle 9: Steueranweisungen für den TSOSLNK

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.

Beispiel 6-1

Statisches Binden zu einem ablauffähigen Programm

```
/START-PROGRAM FROM-FILE = $TSOSLNK ──────────────────────────────────
(1)
 % BLS0500 PROGRAM 'TSOSLNK', VERSION 'V21.0E02' OF '1999-03-15' LOADED
 % BLS0552 ...
*PROG COBOLPROG,LIB=PLAM.LIB,ELEM=COBOLLAD ────────────────────────────
(2)
*INCLUDE COBOLMOD,PLAM.LIB ────────────────────────────────────────────
(3)
*RESOLVE ,$.SYSLNK.CRTE ───────────────────────────────────────────────
(4)
*END ──────────────────────────────────────────────────────────────────
(5)
 % LNK0500 PROG BOUND
 % LNK0506 PROGRAM LIBRARY : PLAM.LIB
 % LNK0507 PHASE WRITTEN TO ELEMENT 'COBOLLAD'
```

(1) Das Dienstprogramm TSOSLNK wird aufgerufen.

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

(3) Die INCLUDE-Anweisung teilt dem Binder mit, dass er das Objektmodul COBOLMOD 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 Handbuch „COBOL2000-Sprachbeschreibung“ [1]) kann der Compiler veranlasst werden, den Maschinencode für eine Übersetzungseinheit 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 definieren (siehe auch Handbuch „TSOSLNK“ [9]):

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 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“ [22] beschrieben.

> **Wichtiger Hinweis:**
> LLMs mit eingebundenem Laufzeitsystem sollten nicht in Bibliotheken abgelegt werden,
>
> - aus denen auch nicht vorgebundene LLMs direkt geladen werden sollen oder
> - die zur Auflösung von Externverweisen mittels AUTOLINK durch den BINDER herangezogen werden.

Beispiel 6-2

Erzeugen eines LLM aus Objektmodulen

```
/START-BINDER ──────────────────────────────────────────────────────────
(1)
%  BND0500 ...
//START-LLM-CREATION INT-NAME=PROG, COPYRIGHT = *NONE ──────────────────
(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
PROCESS
%  BLS0524 LLM 'TESTPROG', VERSION ' ' OF '2006-10-26:14:51:46' 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 potenziell für ILCS vorgesehenen Sprachen. Im Beispiel ist nur die Sprache COBOL2000 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 an Stelle 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 ADD-FILE-LINK-Kommando, z.B.:

```
/ADD-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)
```

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 `programm-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“ [10] 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 „BS2000/OSD-BC Kommandos“ [3].

```
{START-PROGRAM | LOAD-PROGRAM} FROM-FILE =]
*MODULE (LIBRARY={*OMF,ELEMENT=modul | *OMF [,ELEMENT=*ALL] | 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 „Zuweisen von katalogisierten Dateien").

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 entsprechenden ID-Namens in der Übersetzungseinheit. 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 mittelsSET-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 LinknamenBLSLIBnn (00 nn 99) zugewiesen wurden.

Dynamisches Nachladen

Rufen COBOL Module andere externe Unterprogramme über „CALL-bezeichner“, dann sind weitere Bedingungen beim Laden und Starten zu berücksichtigen. Näheres dazu siehe in Kapitel „Programmverknüpfungen".

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 Handbuch „BS2000/OSD-BC Kommandos“ [3]) 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 „Zuweisen von katalogisierten Dateien").
- 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 „BS2000/OSD-BC Kommandos“ [3].

```
{START-PROG | LOAD-PROG} FROM-FILE = {*PHASE(LIB=bibliothek,ELEM=element,
VERS=version) | 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 ReturnCode zugeordnet ist (siehe dazu auch Fehlermeldung COB9119 in Kapitel „Meldungen des COBOL2000-Systems"), wird dieser Return-Code in die letzten beiden Bytes der Rückkehrcode-Anzeige einer überwachenden Jobvariablen (siehe Handbuch [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**	**Auslösen von Spin-Off in Prozeduren** [5]
01**00**	keine	Vom Laufzeitsystem wurde kein Fehler erkannt	--	nein	nein
11**20**	COB9120	Jobvariablen nicht verfügbar	ja		ja
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 inBS2000-Version < 10.0	nein		
11**34**	COB9134	Sort-Fehler	ja		
21**40**	COB9140	fehlerhafte Teilfeldselektion	ja	nein / ja[4]	
21**41**	COB9141	letzte XML-Anweisung noch nicht abgearbeitet	nein		
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	COBOL Laufzeitsystem in CRTE ist inkompatibel zum Objektprogramm	nein		
21**48**	COB9148	CALL oder ADDRESS OF PROGRAMM nicht ausführbar	nein		

21**49**	COB9149	Inkompatible Daten in numerischeditiertem Feld	nein
21**51**	COB9151	Dateien: Nicht abgefangener Ein-/ Ausgabe fehler (keine USE-Prozedur, kein INVALID KEY, kein AT END)	nein
21**52**	COB9152	Verbindung zu Datenbank konnte nicht hergestellt werden	nein
21**53**	COB9153	Fehler beim Konvertieren zwischen EBCDIC und UTF-16	ja
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-Paragrafen	nein
21**59**	COB9159	Fehler beim Verlassen einer XML PROCESSING-Prozedur	nein
21**60**	COB9160	Ablaufeinheit verwendet CANCEL, enthält aber Programme, die mit einem Compiler COBOL85 < 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**65** 21**66** 21**67**	COB9165 COB9166 COB9167	unzulässiger Aufruf bzw. unzulässiges Verlassen von USE Prozeduren	nein
21**68** 21**69** 21**71**	COB9168 COB9169 COB9171	REPORT WRITER: Anwenderfehler	nein nein nein
21**73**	COB9173	SORTlauf nicht erfolgreich	nein
21**74** 21**75**	COB9174 COB9175	Fehlerbehandlung im Programm: Anwenderfehler	nein nein
21**76**	COB9176	REPORT WRITER: Anwenderfehler	nein
21**78**	COB9178	Zu sortierender Satz passt nicht zu SD-Beschreibung	nein
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**82**	COB9182	unzulässige Vererbung bei Klassen bzw. Interfaces	nein
21**84**	COB9184	SORT innerhalb der SORT-Steuerung	nein
21**85**	COB9185	Fehler im Zusammenhang mit OO-Sprachmittel	nein
21**88**	COB9188	XML-Parser nicht gefunden	nein

21**89**	COB9189	PARTIAL-BIND-Laufzeitsystem nicht gefunden	nein	
31**91**	COB9191	SUPER-Klasse nicht gefunden	nein	ja
31**92**	COB9192	Programmende wurde erreicht, ohne dass STOP RUN oder EXIT PROGRAM ausgeführt wurde	nein	
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**	COB9198	Hardware-Unterbrechung	nein	
31**99**	keine	WROUT-Fehler: Es kann keine Meldung mehr ausge geben werden	nein	

Tabelle 10: 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 „Meldungen des COBOL2000-Systems"

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 programmüberwachende Jobvariable gesetzt.

4) Stapelbetrieb: nein
Dialogbetrieb: Abfrage ja/nein

5) Wenn Spin-Off ausgelöst ist, werden alle nachfolgenden Kommandos mit Ausnahme der Kommandos SET-JOB-STEP, EXIT-JOB, LOGOFF, CANCEL-PROCEDURE, END-PROCEDURE und EXIT-PROCEDURE ignoriert. Dabei beendet das Kommando SET-JOB-STEP den Spin-Off und die Verarbeitung wird mit dem nächsten Kommando fortgesetzt.

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 Handbuch [11]) in einer PLAM-Bibliothek abgelegt werden.

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 COBOL2000-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“:

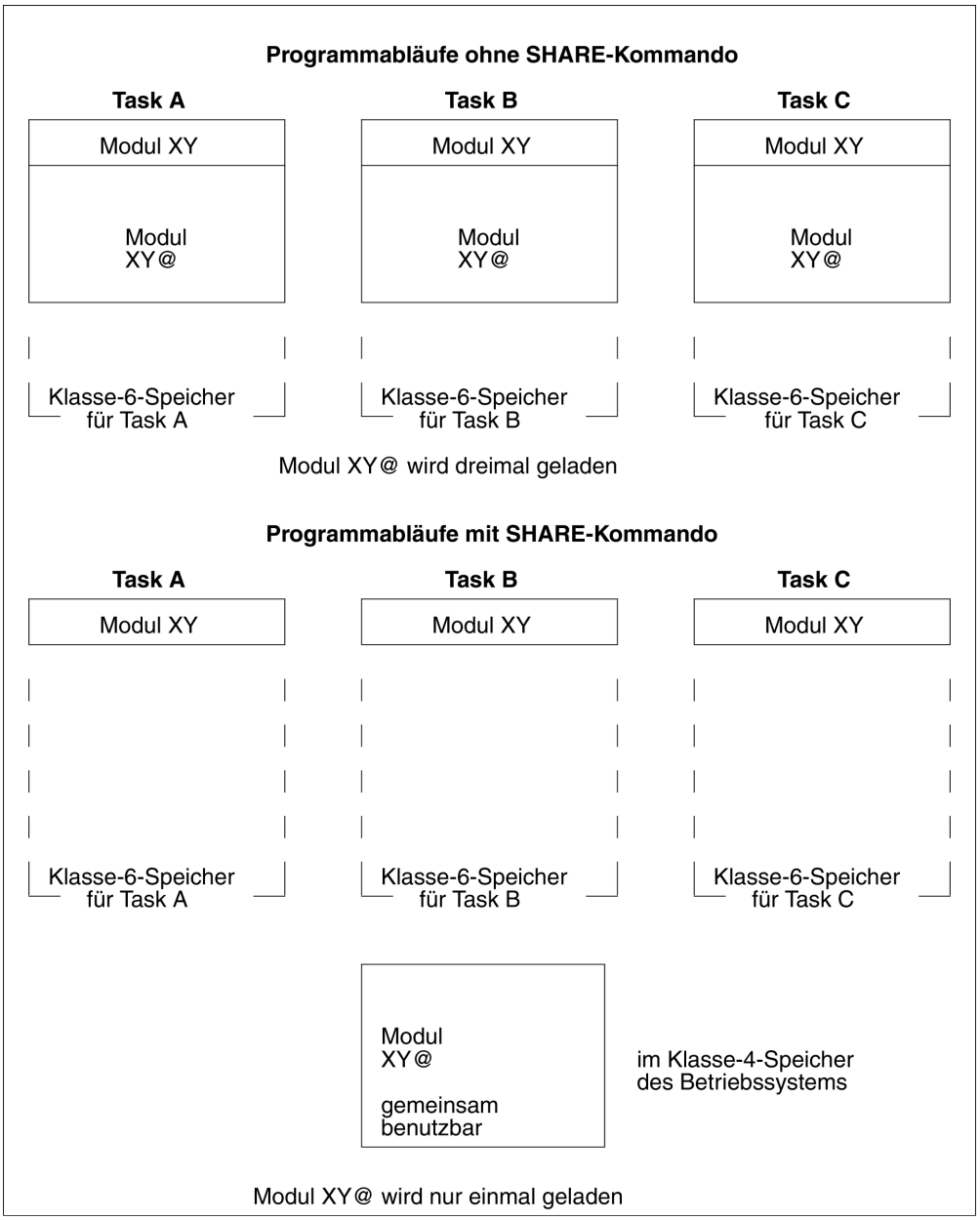


Bild 3: Shared Code

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 der Übersetzungseinheit Testhilfezeilen einbauen und sie bei Bedarf aktivieren. Dies setzt voraus, dass schon bei der Erstellung der Übersetzungseinheit mögliche Fehlersituationen eingeplant werden. Die Diagnose unvorhergesehener Fehler kann es daher erforderlich machen, Testhilfezeilen abzuändern oder hinzuzufügen und anschließend die Übersetzungseinheit neu zu übersetzen. Testhilfezeilen werden im Handbuch „COBOL2000-Sprachbeschreibung“ [1] und in Abschnitt „Auftrags- und Benutzerschalter" beschrieben.

Die Testhilfen lassen sich im POSIX-Subsystem analog verwenden (siehe Kapitel „COBOL2000 und POSIX").

7.1 Dialogtesthilfe AID

In COBOL2000-BC nicht unterstützt !

In diesem Benutzerhandbuch soll AID lediglich kurz vorgestellt werden. Die ausführliche Beschreibung dieser Testhilfe befindet sich in den Handbüchern [8], [20] und [21]. Kenntnisse aus dem Handbuch „AID“ [8] werden vorausgesetzt.

AID zeichnet sich durch folgende Leistungsmerkmale aus:

1. Es bietet die Möglichkeit, „symbolisch“ zu testen, d.h. in den Kommandos an Stelle absoluter Adressen auch symbolische Namen aus der Übersetzungseinheit anzugeben, wenn die dafür nötigen LSD-Informationen beim Übersetzen erzeugt und später an das geladene Programm weitergegeben werden (siehe Abschnitt „Symbolisches Testen mit AID").
   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 (so genannte Subkommandos) ausführen zu lassen,
   - nach einer Programmunterbrechung ein Kapitel oder einen Paragrafen der PROCEDURE DIVISION zu vereinbaren, mit dem - abweichend von der codierten Programmlogik - der Testablauf fortgesetzt werden soll (%JUMP-Anweisung (siehe Handbuch „AID“ [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 „TEST-SUPPORT-Option" bzw. „Tabelle der COMOPT-Operanden"),
   - sich die Inhalte von Feldern in einer Form ausgeben zu lassen, welche die Datendefinitionen der Übersetzungseinheit 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 Paragrafen

mit den in der Übersetzungseinheit 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,

- 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 COBOL2000
- 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 COBOL2000[2)]	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 11: Operanden zur Erzeugung von LSD-Informationen

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

[2] Bei Verwendung der COMOPT GEN-SHARE=YES bzw. der SDF-Option SHARE-CODE=YES werden beim Debuggen für den Trace nur Statements aus dem Code- oder Datenmodul aufgelistet.

7.1.2 Symbolisches Testen mit AID

Beim symbolischen Testen mit AID können Datenfelder, Übersetzungseinheiten, Kapitel und Paragrafen mit den Namen angesprochen werden, die im Quelltext 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 Paragrafennamen) bzw.
- S’nverbm’ (für eine Zeile mit COBOL-Verben)

angeben. Einen solchen **LSD-Namen** bildet COBOL2000 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 COBOL2000 bei der Übersetzung vergeben hat. Sie muss ohne führende Nullen angegeben werden. Soll als Zeilennummer die (maximal sechsstellige) Folgenummer der Übersetzungseinheit 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 ein- oder zweistellige 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 Handbuch „AID“ [8].

Liste der COBOL-Verben und ihrer Abkürzungen:

ACC	ACCEPT	INI	INITIATE
ADD	ADD	INSP	INSPECT
ADDC	ADD CORRESPONDING	INV	INVOKE
ALLO	ALLOCATE	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	RAIS	RAISE
DSC	DISCONNECT	REA	READ
END	END-xxx[1] [2]	REDY	READY
ENTR	ENTRY	REL	RELEASE
ERA	ERASE	RES	RESUME
EVAL	EVALUATE	RET	RETURN
EXI	EXIT	REW	REWRITE
EXI	EXIT PARAGRAPH	SEA	SEARCH
EXI	EXIT SECTION	SET	SET
EXIT	EXIT METHOD	SOR	SORT
EXIT	EXIT PROGRAM	STA	START
FET	FETCH	STO	STOP
FIN	FINISH	STOR	STORE
FND	FIND	STRG	STRING
FRE	FREE	SUB	SUBTRACT
GEN	GENERATE	SUBC	SUBTRACT CORRESPONDING
GET	GET	TER	TERMINATE
GO	GOBACK	UNST	UNSTRING
GOT	GO TO	WRI	WRITE
IF	IF	XML	XML
INIT	INITIALIZE		

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.

7.1.3 Vordefinierte Informationen

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`	ID-Name des Objekts
`_EBCDIC_CCSN`	Name der EBCDIC Variante, die bei Konvertierungen zwischen alphanumerischen und nationalen Daten angenommen wird

Informationen über den Ausnahmezustand

Mit dem AID-Kommando

```
%D[ISPLAY] _LAST_EXCEPTION
```

können allgemeine Informationen über den letzten Ausnahmezustand angefordert werden.

Format der Ausgabe:

```
01 _LAST-EXCEPTION.
   02 _EXCEPTION_NAME PIC X(31).
```

`_EXCEPTION_NAME`	Name der Ausnahmesituation, die zur Auslösung des Ausnahmezustands geführt hat (Leerzeichen, wenn kein Ausnahmezustand existiert).

7.1.4 Hinweise zum symbolischen Testen von geschachtelten Programmen

- Setzen von Testpunkten
  - Paragrafen und Kapitel des inneren Programms, in dem die Unterbrechungsstelle liegt, können ohne Qualifikation angesprochen werden.
  - Auf Kapitel und Paragrafen 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

    Dieses Vorgehen ist nur dann sinnvoll, wenn program-id nicht länger als 8 Zeichen ist oder ein LLM generiert wurde, da sonst der Source-Name, nicht aber der Procedure-Name auf 8 Zeichen abgeschnitten wird.
  - 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 Paragrafen, gelegentlich zusätzliche LABEL-Angaben.

7.1.5 Hinweise zum Testen von objektorientierten Programmen

- Adressierung
  - **Klassen** werden durch eine Source-Qualifizierung angesprochen: S=<class>. <class> ist der Name, der im CLASS-ID Paragrafen angegeben ist.
  - **Methoden** werden durch eine Procedure-Qualifizierung angesprochen:
    PROC={FACTORY | OBJECT}.PROC=<method>, wobei <method> der Name ist, der im METHOD-ID Paragrafen angegeben ist.

    Eine Source-Qualifizierung ist dann notwendig, wenn der aktuelle Programmpunkt nicht in (einer Methode) der Klasse liegt.
    Procedure-Qualifizierungen sind nur soweit nötig, wie dies für die eindeutige Identifizierung erforderlich ist. So kann PROC={FACTORY | OBJECT} für Methoden grundsätzlich entfallen, da der Methodenname in der Klasse eindeutig sein muss.
- Kommandos
  - ***Setzen von Testpunkten***

    Das Setzen von Testpunkten in Methoden ist mit der Source- und Procedure-Qualifikation möglich:
    %INSERT [S=<class>.] [PROC=<method>.] srcref

    Auf eine Objektreferenz kann eine Schreibüberwachung gesetzt werden:
    %ON %WRITE(objref). Die Anzeige einer durch NEW veränderten Objektreferenz ist aber erst nach Rückkehr an die Aufrufstelle möglich.
  - ***Ablaufverfolgung***

    Bei %TRACE können Klassen und Methoden als Trace-Bereich angegeben werden:
    %TRACE <n> IN S=<class>.[PROC={FACTORY | OBJECT}.PROC=<method>]
  - ***Anzeigen von Daten***

    %DISPLAY

    Daten eines Objektes sind nur sichtbar, wenn sich die Unterbrechungsstelle in einer Methode dieses Objektes befindet. In diesem Fall wird keine Qualifikation angegeben.

    Daten in einer Methode sind nur innerhalb dieser Methode sichtbar.

    Eine Objektreferenz wird wie folgt angezeigt:

    ```
    <level> objref
             <level+1> FACTORY | OBJECT | NULL
             <level+1> class-name
    ```

    Die erste Komponente gibt an, ob die Referenz auf das Factory-Objekt oder ein normales Objekt verweist oder eine Nullreferenz ist. Die zweite Komponente zeigt den Namen der Klasse des aktuell referenzierten Objektes an; für eine Nullreferenz entfällt sie.

    %SD

    %SD zeigt die Daten in der aktuellen dynamischen Aufruf-Verschachtelung von Programmen und Methoden an. Für Methoden werden nur die lokalen Daten der Methode, nicht aber die Daten des umgebenden Objektes ausgegeben.
    Zusätzlich werden pro Klasse source-modul-globale Daten, wie z.B _COMPILATION_DATE ausgegeben.
  - ***Ändern von Daten***

    %SET, %MOVE

Eine high-level Zuweisung an eine Objektreferenz wird von AID mit einer Fehlermeldung (Types are not convertible...) abgewiesen. Ein low-level Zugriff auf Objektreferenzen ist möglich, liegt aber vollständig in der Verantwortung des Benutzers.

7.1.6 Hinweise zum Testen von Programmen mit benutzerdefinierten Typen

AID V3.1A unterstützt die TYPEDEF-Klausel sowie typbezogene Zeiger in COBOL2000.

Ein Dereferenzierungs-Operator und ein Adress-Operator ergänzen nun die bisher gängigen AID-Operatoren (siehe Handbuch „AID“ [8]).

Der Dereferenzierungs-Operator dient dem Zugriff auf das durch einen Zeiger adressierte Datum. Er wird durch einen Stern dargestellt und kann mit der COBOL-Qualifikation (IN, OF) und der COBOL-Subskribierung kombiniert werden.

Der Adressoperator liefert die Adresse eines Datums zur Versorgung eines Zeigers oder zur weiteren Verwendung in Low Level AID. Hierfür unterstützt AID die COBOL-Syntax ADDRESS OF.

- **Zugriff auf Datennamen**

  Für die Datennamen der TYPEDEF-Klausel gibt es innerhalb der AID-Kommandos keine Verwendung.

  - Die Eingabe einfacher, qualifizierter und indizierter Symbole erfolgt in AID genauso wie in COBOL. Das bedeutet, dass insbesondere Teilqualifizierungen erlaubt sind, solange sie eindeutig sind. Aus Gründen der Performance ist es für Programme mit sehr vielen Gruppen (Strukturen) empfehlenswert, ein Eingabesymbol vollständig zu qualifizieren. Dadurch wird der Suchprozess beschleunigt und der Eindeutigkeitstest kann entfallen. Für Symbole mit einer dereferenzierten Komponente (z.B. %D NAME IN *ADDRESS-START) wird stets die vollständige Qualifizierung vorausgesetzt. Zudem sind Indexangaben stets exakt anzugeben.
  - Für komplexe Datenzugriffe, die durch Zusammensetzung von Qualifizierung, Subskribierung und Dereferenzierung entstehen, gelten folgende Regeln:
    - Die Abarbeitung erfolgt stets von rechts nach links. Der am weitesten rechts liegende Operator wird zuerst abgearbeitet.
    - Ist ein Operand mit einem Operator geklammert, so hat der geklammerte Operator Vorrang in der Abarbeitung von rechts nach links.
- **Adress-Selektor**

  Genau wie in COBOL ist das Schlüsselwort des Adress-Selektors **ADDRESS OF**. Es ist reserviert und gilt **nicht** bei der Einstellung %AID SYMCHARS=NOSTD.
- **Zuweisungen und Vergleiche**
  - Zuweisungen und Vergleiche von Variablen mit der gleichen TYPEDEF-Klausel ohne STRONG-Angabe können analog zu Gruppen nur auf Low Level Ebene, d.h. durch explizite Konvertierung der Gruppen in hexadezimalen Strings, durchgeführt werden.
  - Bei Zuweisungen und Vergleichen von Variablen mit der gleichen TYPEDEF-Klausel mit der STRONG-Angabe kann auf die explizite Konvertierung in hexadezimalen Strings in der AID-Kommando-Eingabe verzichtet werden. AID prüft, ob Quelle und Ziel die gleiche[1] TYPEDEF-Klausel mit der STRONG-Angabe besitzen und führt dann die Zuweisung oder den Vergleich aus. Bei der Ausführung wird jedoch intern die String-Konvertierung auf die Variable als Ganzes und nicht auf die einzelnen Komponenten durchgeführt.
  - Bei Zuweisungen und Vergleichen von typbezogenen Zeigern wird überprüft, ob die Zeiger den gleichen Referenztyp haben. Wenn der Referenztyp eine Gruppe (Struktur) mit der TYPEDEF-Klausel ist, ist zudem die STRONG-Angabe in der Deklaration des Typs notwendig. Wird dem Zeiger eine Adresse über den Adress-Selektor zugewiesen oder wird ein Zeiger mit einem Adress-Selektor verglichen, so findet die analoge Typüberprüfung zwischen Referenztyp des Zeigers und dem Argumenttyp des Adress-Selektors statt.

Beispiel 7-2

```
01 PT-TYP TYPEDEF USAGE POINTER TO PERSON.
01 PERSON TYPEDEF STRONG.
   02 NAME PIC X(30).
   02 VORNAME PIC X(30).
01 VERWEIS TYPE PT-TYP.
01 PERS1 TYPE PERSON.
01 PERS2 TYPE PERSON.
```

Mögliche Eingabe in AID:

```
%SET ADDRESS OF PERS1 INTO VERWEIS.
%D *VERWEIS
%D NAME IN *VERWEIS                              (1)
%D ADDRESS OF PERS1                              (2)
%SET PERS1 INTO PERS2                            (3)
```

(1) Zeigt den Inhalt des Datenfeldes, auf welches *VERWEIS* verweist als Gruppe vom Typ *PERSON* bzw. das elementare Feld *NAME* in dieser Gruppe.

(2) Gibt sedezimal die Adresse von *PERS1* zur weiteren Nutzung in Low Level AID aus.

(3) Entspricht in COBOL: MOVE PERS1 TO PERS2.

[1] Beachten Sie: Äquivalente Typen behandelt AID nicht als „gleiche“ Typen (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]).

7.2 Testhilfezeilen

Auf Übersetzungseinheit-Ebene bietet COBOL2000 für die Diagnose von logischen Fehlern Testhilfezeilen an. Dabei handelt es sich um besonders gekennzeichnete Zeilen in der Übersetzungseinheit, die

- lediglich COBOL-Anweisungen für Testzwecke enthalten und
- bei der Übersetzung nach Bedarf als Anweisungs- oder als Kommentarzeilen behandelt werden können.

COBOL2000 unterstützt die Anwendung von Testhilfezeilen durch folgende Sprachmittel (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]):

- Die WITH DEBUGGING MODE-Klausel im SOURCE-COMPUTER-Paragrafen 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 in der Übersetzungseinheit 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:
  - In der Übersetzungseinheit sind Testhilfezeilen erst nach dem OBJECT-COMPUTER-Paragrafen erlaubt.
  - Die COBOL-Übersetzungseinheit muss sowohl mit als auch ohne Berücksichtigung der Testhilfezeilen syntaktisch korrekt sein.
  - Testhilfezeilen sind nur im Fixed-Format erlaubt.

8 Schnittstelle zwischen COBOL-Programmen und BS2000/OSD

Die Schnittstelle zwischen COBOL-Programmen und dem POSIX-Subsystem ist in Kapitel „COBOL2000 und POSIX" 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 COBOL2000 durch folgende Sprachmittel (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]):

- Die Vereinbarung programminterner Merknamen für Systemdateien im SPECIAL-NAMES-Paragrafen 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 th 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-Paragrafen) 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 Abschnitt „Systemdateien: Primärzuweisungen, Umweisungen, Satzformate"), 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-Paragrafen) 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 14 im Abschnitt „Systemdateien: Primärzuweisungen, Umweisungen, Satzformate"), 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 **Drucker** ausgegeben und anschließend gelöscht werden.	
SYSOPT	temporäre SPOOLOUT-Datei (EAM-Datei), die bei Task-Ende auf **Diskette** ausgegeben und anschließend gelöscht wird.	

Tabelle 12: 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 im Handbuch „BS2000/OSD-BC Kommandos“ [3].

Systemdatei	**Umweisung auf**	**mit dem Kommando**
SYSDTA	katalog. Plattendatei (SAM oder ISAM) oder PLAM-Bibliothek	ASSIGN-SYSDTA dateiname ASSIGN-SYSDTA *LIBRARY(bibliothek, element)
	Diskette	ASSIGN-SYSDTA *DISKETTE(...)
SYSIPT	katalog. Plattendatei (SAM oder ISAM)	ASSIGN-SYSIPT dateiname
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 13: 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,V	maximal 80 Byte Daten, falls SAM-Datei maximal 80 Byte: 72 Byte Daten, falls ISAM-Datei; die Bytes 73-80 enthalten den ISAM-Schlüssel
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 14: Satzformate und Satzlängen der Systemdateien

8.2 Auftrags- und Benutzerschalter

Das BS2000 stellt jedem Auftrag (Task) 32 Auftragsschalter (nummeriert von 0 bis 31) und jeder Benutzerkennung 32 Benutzerschalter (nummeriert von 0 bis 31) zur Verfügung (siehe Handbuch [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 COBOL2000 durch folgende Sprachmittel (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]):

- Die Vereinbarung programminterner Merknamen für Auftrags- und Benutzerschalter und ihre Zustände im SPECIAL-NAMES-Paragrafen 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-Paragrafen vereinbarten Bedingungsnamen von Schalterzuständen enthalten und sie auf diese Weise für die Steuerung des Programmablaufs auswerten.
  - SET (Format 3; siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]) kann über die im SPECIAL-NAMES-Paragrafen vereinbarten Merknamen auf Schalter zugreifen und ihre Zustände verändern.

Beispiel 8-2

Verwendung von Auftragsschaltern

Im folgenden Ausschnitt aus einem Dialogauftrag sieht eine 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 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)
... /CALL-PROC PROG.SYSTEM
```

```
Die Datei PROG.SYSTEM enthält  ———————————————————— (2)
folgende Kommandos:
/BEGIN-PROC ...
  ...
/START-PROGRAM PROG-1 ———————————————————————————— (3)
```

```
  Ausschnitt aus PROG-1:
  ...
  SPECIAL-NAMES.                              (4)
  TSW-12 IS SWITCH-12                          |
      ON IS ON-12                              |
  TSW-13 IS SWITCH-13                          |
      ON IS ON-13                             (4)
  ...
  PROCEDURE DIVISION.
  ...
      IF ON-12 PERFORM A-PAR. ———————————— (5)
      PERFORM B-PAR.
      ...
      IF FIELD = 99 SET SWITCH-13 TO ON. ——— (6)
      STOP RUN.
  A-PAR.
      ...
  B-PAR.
      ...
```

```
...
/SKIP-COMMANDS TO-LABEL .END,IF=JOB-SWITCHES (OFF=13) ——— (7)
/START-PROGRAM PROG-2
/.END 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 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.

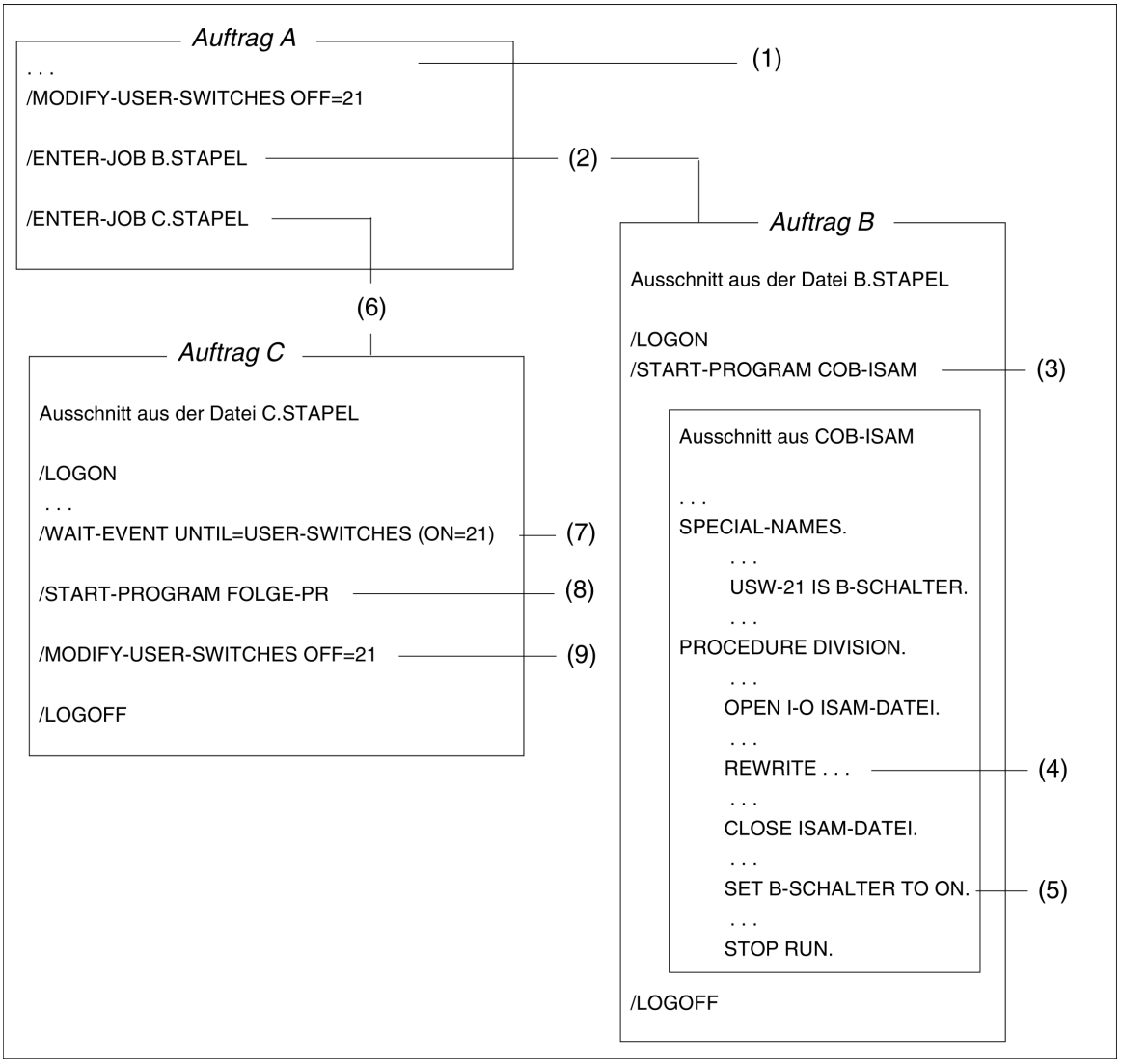
(7) Die Prozedur wertet den Status des Auftragsschalters 13 aus: Falls er vonPROG-1 nicht auf ON gesetzt wurde, verzweigt sie zum Ende, andernfalls führt sie zusätzlich zu PROG-1 das Programm PROG-2 aus.

(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 aktualisierteISAM-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 COBOL2000 durch folgende Sprachmittel (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]):

- Die Vereinbarung von Linknamen und programminternen Merknamen für Jobvariablen im SPECIAL-NAMES-Paragrafen 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-Paragrafen) 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-Paragrafen) 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 Tabelle10 in Abschnitt „Programmbeendigung").

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=UPDATE,JV-NAME=CONTROL.RUN ———————————— (1)
/START-PROGRAM PROG.WORK-1
```

```
Programmausschnitt:
...
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T
    JV-UPDATE IS FIELDJV. ———————————————————————————— (2)
     ...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DATE-TODAY      PIC X(6). ———————————————————————— (3)
01  CONTENTS-JV. ———————————————————————————————————— (4)
  05  DATE-UPDATE   PIC X(6).
  05  FILLER        PIC X(20).
  05  NUM-UPDATE    PIC 9(4).
     ...
PROCEDURE DIVISION.
    ACCEPT CONTENTS-JV FROM FELDJV. ——————————————————— (5)
    ACCEPT DATE-TODAY FROM DATE.
    IF DATE-UPDATE NOT EQUAL DATE-TODAY ——————————————— (6)
      PERFORM WORK
        ELSE PERFORM ALREADY-UPDATED.
     ...
WORK.
     ...
    MOVE DATE-TODAY TO AKT-DAT.                          (7)
    ADD 1 TO NUM-UPDATE.                                  |
    DISPLAY CONTENTS-JV UPON FIELDJV.                    (7)
     ...
ALREADY-UPDATED.
    DISPLAY "END OF UPDATE"
      UPON T.
     ...
```

```
/SHOW-JV JV-NAME(CONTROL.RUN) ———————————————————————————— (8)
%930629 UPDATE 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-Paragrafen 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 S-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 der Übersetzungseinheit
- die seit dem LOGON 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 COBOL2000 durch folgende Sprachmittel:

- Die Vereinbarung programminterner Merknamen für die einzelnen Informationsarten im SPECIAL-NAMES-Paragrafen 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-Paragrafen) 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 (`COBOL2000 'BLANK',COBOL2000B,COBOL2000R`)
11-20	Version des Compilers Format: Vzz.zbzzzz z = Ziffer oder Leerzeichen b = Buchstabe oder Leerzeichen, (z.B. „V01.0A “)
21-30	Datum der Übersetzung Format: JJJJ-MM-TT (z.B. „1999-12-31“)
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 TaskInhalt: U für BenutzerS für Systemverwalter
23-32	Betriebssystemversion Format: Vzz.zbzzzz (z.B. „V11.2 “)
33-40	Name des nächsten Rechners, an den die Datensichtstation angeschlossen ist

Tabelle 15: Struktur der Compiler- und Betriebssysteminformationen

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 werdenkönnen, 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'BLANK'

Tabelle 15: 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  PROGRAM-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 X(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 „COBOL2000 und POSIX" beschrieben.

9.1 Grundsätzliches zum Aufbau und zur Verarbeitung katalogisierter Dateien

In diesem Kapitel werden folgende Themen behandelt:

- Grundbegriffe zum Aufbau von Dateien
- Zuweisen von katalogisierten Dateien
- Festlegen von Dateimerkmalen
- Platten- und Dateiformate

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 DVSträglich

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 sequenzielle, relative und indizierte Dateiorganisation. Die Möglichkeiten und Besonderheiten der einzelnen Organisationsformen werden in den Abschnitten „Sequenzielle Dateiorganisation", „Relative Dateiorganisation" und „Indizierte Dateiorganisation" näher erläutert. Jeder dieser Organisationsformen entspricht eine Zugriffsmethode des DVS. Die Zuordnung kann der folgenden Tabelle entnommen werden:

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

Tabelle 16: 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 Handbuch „COBOL2000-Sprachbeschreibung“ [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 ADD-FILE-LINK-Kommando bzw. mittels der Programmangaben verwendet werden.

Ein Wert für die Puffergröße wird vom Compiler bei der Übersetzung aus den Angaben in der Übersetzungseinheit über Satz- und Blocklänge für jede Datei berechnet. Diese Voreinstellung kann bei der Zuweisung der Datei durch die Angabe des BUFFER-LENGTH-Operanden im ADD-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 Abschnitt „Platten- und Dateiformate").

Außer bei neu angelegten Dateien (OPEN OUTPUT) hat die im Katalog eingetragene Blockgröße stets Vorrang gegenüber den Blockgrößenangaben im Programm bzw. im ADD-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 Handbuch „COBOL2000-Sprachbeschreibung“ [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 ADD-FILE-LINK- bzw. ein ASSIGN-systemdatei-Kommando herstellen. Welches der beiden Kommandos zu verwenden ist, hängt vom Eintrag in der ASSIGN-Klausel (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [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 ADD-FILE-LINK-Kommando

Die Zuweisung über das ADD-FILE-LINK-Kommando kann 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 ADD-FILE-LINK-Kommando absetzen, in dessen LINK-NAME-Operanden er den vereinbarten Linknamen angibt. Mit Hilfe weiterer Operanden des ADD-FILE-LINK-Kommandos können damit zugleich auch Dateimerkmale festgelegt werden.

Jeder Linkname muss den Anforderungen des BS2000 an einen Linknamen genügen (siehe dazu Handbuch [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 ADD-FILE-LINK-Kommando

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

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

Bei COBOL-Programmen mit SORT (siehe Kapitel „Sortieren und Mischen") 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

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 ADD-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 Handbuch [3]).

Beispiel 9-2

Änderung von Dateizuweisungen

```
/ADD-FILE-LINK INOUTFIL,FILE.UPDATE.1 ————————————————————————(1)
/START-PROGRAM AKTUELL
   ...
/ADD-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 ADD-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.

Implizite Zuweisung über Voreinstellungen

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.COBOL.linkname, die es vorher angelegt hat.
  Bei ENABLE-UFS-ACCESS = YES schreibt das Programm unmittelbar in die DateiFILE.COBOL.linkname.
- Bei einer Eingabedatei, deren SELECT-Klausel die Angabe OPTIONAL enthält, 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.

Zuweisung über das ASSIGN-*systemdatei*-Kommando

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

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-Paragrafen des COBOL-Programms LISTPROG:	`SELECT DRUCK-DATEI ASSIGN TO PRINTER.`
Zuweisung der Datei LIST.DATEI und Programmaufruf:	`/ASSIGN-SYSLST LIST.DATEI` `/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

Für das Einrichten von Dateien steht im BS2000 das CREATE-FILE Kommando zur Verfügung. Ein Task File Table-Eintrag mit weiteren Dateimerkmalen wird später mit dem ADD-FILE-LINK-Kommando erzeugt. Die vollständigen Formate und eine ausführliche Beschreibung können in den Handbüchern [3] oder [4] nachgelesen werden.

Katalogeintrag

Beim Festlegen der Dateimerkmale mit dem CREATE-FILE Kommando ist darauf zu achten, dass eine ausreichende Anfangszuweisung von Speicherplatz erfolgt.
Dies gilt insbesondere für Dateien, bei denen die Blockgröße die vom Betriebssystem standardmäßig angenommene Anfangszuweisung übersteigt. Die Blockgröße errechnet sich aus den Angaben im COBOL-Programm (siehe „COBOL2000-Sprachbeschreibung“ [1]). Ergibt sich aus dem COBOL Programm eine Blocklänge (STD, n), sollte die PRIMARY-ALLOCATION im SPACE-Operand des CREATE-FILE Kommandos

- für sequentielle Dateien mindestens 2n sein.
- für indizierte und relative Dateien mindestens 2n+2 sein.

Task File Table

Zu jeder Datei, für die ein ADD-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 ADD-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 ADD-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 Abschnitt „Zuweisen von katalogisierten Dateien"). 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 ADD-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 ADD-FILE-LINK-Kommando als Nulloperanden vereinbart wurden.

Beim Dateizugriff kann dieses Verfahren insbesondere dann zu Konflikten führen, wenn im ADD-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	ADD-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	ADD-FILE-LINK-Kommando
ASSIGN-KlauselORGANIZATION-Klausel	ACCESS-METHOD-Operand

- Widersprüchliche Angaben zum **Satzformat**

COBOL-Programm	ADD-FILE-LINK-Kommando
RECORD-KlauselRECORDING MODE-Klausel	RECORD-FORMAT-Operand

- Widersprüchliche Angaben zur **Satzlänge**

COBOL-Programm	ADD-FILE-LINK-Kommando
RECORD-KlauselDatensatzerklärung	RECORD-SIZE-Operand

- Widersprüchliche Angaben zum **Satzschlüssel**

COBOL-Programm	ADD-FILE-LINK-Kommando
RECORD KEY-KlauselDatensatzerklärung	KEY-POSITION-OperandKEY-LENGTH-Operand

- Widersprüchliche Angaben zum **Plattenformat** oder **Dateiformat**

Katalogeintrag	ADD-FILE-LINK-Kommando
BLK-CONTR =BUF-LEN =	BLOCK-CONTROL-INFO-OperandBUFFER-LENGTH-Operand

Beispiel 9-4

Erzeugen und Abbilden eines TFT-Eintrags

(Abbildung in BS2000/OSD V5.0)

```
/ADD-FILE-LINK INOUTFIL,ISAM.UPDATE, -
(1)
/              BUFFER-LENGTH=*BY-CATALOG,
|
               SUPPORT=*DISK(SHARED-UPDATE=*YES)
(1)
/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
————————————————————— VOLUME
————————————————————————————————————————————————————————
DEV-TYPE   = *NONE       T-SET-NAME = *NONE
VSN/DEV    = PUBN03/D3480
```

(1) Das ADD-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 (siehe Abschnitt „Simultanverarbeitung von Dateien (SHARED-UPDATE)").

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 ADD-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).

NK-Datenträger werden 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.

Bei Verwendung von NK4-Datenträgern muss gewährleistet sein, dass die Satzlängen einer geradzahligen Blockung entsprechen.

Das Blockformat für eine COBOL-Datei lässt sich mit dem BLOCK-CONTROL-INFO-Operanden des ADD-FILE-LINK-Kommandos bestimmen:

```
/ADD-FILE-LINK ...,-
/   BLOCK-CONTROL-INFO = BY-PROGRAM / BY-CATALOG / WITHIN-DATA-BLOCK /PAMKEY / NO
```

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 „Einführung in das DVS“ [4].

Werden im BLOCK-CONTROL-INFO- oder im BUFFER-LENGTH-Operanden des ADD-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 ADD-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 Blockes.

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 17: 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:

```
/ADD-FILE-LINK ...,RECORD-FORMAT=FIXED,BUFFER-LENGTH=STD(SIZE=2), -
/                                      BLOCK-CONTROL-INFO=WITHIN-DATA-BLOCK
```

maximale Satzlänge (nach Formel in Tabelle 17):
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 18: 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 Sequenzielle Dateiorganisation

Es gibt zwei Arten sequenziell organisierter Dateien: satzsequenzielle und zeilensequenzielle Dateien. Die folgende allgemeine Beschreibung bezieht sich auf satzsequenziell organisierte Dateien.
Die Abweichungen und Einschränkungen der zeilensequenziellen Organisation gegenüber der satzsequenziellen Organisation sind in Abschnitt „Zeilensequenzielle Dateien" beschrieben.

9.2.1 Merkmale sequenzieller Dateiorganisation

Die Sätze einer sequenziell 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 sequenziellen Datei

- Sätze einzufügen,
- Sätze zu löschen oder
- die Position eines Satzes innerhalb der festgelegten Reihenfolge zu verändern.

Sequenzielle 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 sequenzieller Dateien verwenden COBOL-Programme die Zugriffsmethode SAM des DVS. Einzelheiten darüber können im Handbuch [4] nachgelesen werden.

Sequenzielle Dateien können sowohl auf Magnetbändern als auch auf Geräten mit direktem Zugriff (Plattenspeichern) eingerichtet werden.

9.2.2 COBOL-Sprachmittel für die Verarbeitung sequenzieller Dateien

Das folgende Programmskelett gibt einen Überblick über die wichtigsten Klauseln und Anweisungen, die COBOL2000 für die Verarbeitung sequenzieller 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 in der Übersetzungseinheit 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 ADD-FILE-LINK-Kommando angefordert, im Stapelbetrieb die AT END-Bedingung ausgelöst,
- bei OPEN I-O oder OPEN EXTEND eine Datei mit dem Namen FILE.COBOL.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 Handbuch „COBOL2000-Sprachbeschreibung“ [1]).

`ORGANIZATION IS SEQUENTIAL`

legt fest, dass die Datei satzsequenziell organisiert ist.

Die ORGANIZATION-Klausel kann bei satzsequenziellen Dateien entfallen, da satzsequenzielle Dateiorganisation die Standardannahme des Compilers ist.

`ACCESS MODE IS SEQUENTIAL`

bestimmt, dass auf die Sätze der Datei nur sequenziell zugegriffen werden kann. Die ACCESS MODE-Klausel ist optional und dient bei sequenziellen Dateien lediglich der Dokumentation, da sequenzieller Zugriff die Standardannahme des Compilers und die einzige für sequenzielle 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 „Verarbeiten von Magnetbanddateien" 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 BLOCKCONTAINS-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 in der Übersetzungseinheit 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 ADD-FILE-LINK-Kommando verändert werden (siehe Abschnitt „Festlegen von Dateimerkmalen"), 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 Abschnitt „Platten- und Dateiformate").

Außer bei neu angelegten Dateien (OPEN OUTPUT) hat die im Katalog eingetragene Blockgröße stets Vorrang gegenüber den Blockgrößenangaben im Programm bzw. im ADD-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 COBOL2000 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 Abschnitt „Zulässige Satzformate und Zugriffsarten").

`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 derRECORD-Klausel festgelegt werden (siehe Abschnitt „Zulässige Satzformate und Zugriffsarten").

```
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 Abschnitt „Eröffnungsarten und Verarbeitungsformen (sequenzielle Dateien)").

```
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 Abschnitt „Eröffnungsarten und Verarbeitungsformen (sequenzielle Dateien)" 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

Sequenzielle 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.

In der COBOL-Übersetzungseinheit wird das Format der zu verarbeitenden Sätze in der RECORD- oder der RECORDING MODE-Klausel festgelegt (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]). Welche Angaben dabei dem jeweiligen Satzformat zugeordnet sind, ist in der folgenden Tabelle zusammengestellt:

	Angabe in der	
Satzformat	**RECORD-Klausel**	**RECORDING MODE-Klausel**
feste Länge	RECORD CONTAINS...CHARACTERS (Format 1)	
variable Länge	RECORD IS VARYING IN SIZE... (Format 2) oder (Format 3) RECORD CONTAINS...TO...	
undefinierte Länge	Vereinbarung mit der RECORD-Klauselnicht möglich	RECORDING MODE IS U

Tabelle 19: 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 sequenziellen Datei kann nur sequenziell 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.

In der COBOL-Übersetzungseinheit wird die Zugriffsart durch die ACCESS MODE-Klausel festgelegt; für sequenzielle 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 (sequenzielle Dateien)

Mit den Sprachmitteln eines COBOL-Programms lassen sich sequenzielle 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:

```
WRITE... [FROM...] [{BEFORE | AFTER} ...]
         [AT END-OF-PAGE...]
         [NOT AT END-OF-PAGE...]
         [END-WRITE]
```

In diesem Modus können sequenzielle 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 „Zeilensequenzielle Dateien" 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 sequenzielle 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:

```
WRITE...[FROM...] [{BEFORE | AFTER} ...]
        [AT END-OF-PAGE...]
        [NOT AT END-OF-PAGE...]
        [END-WRITE]
```

In diesem Modus können am Ende einer sequenziellen 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 sequenziellen 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 Zeilensequenzielle Dateien

Die zeilensequenzielle Organisation von COBOL-Dateien ist ein Sprachmittel des X/OpenStandards. Das entsprechende Sprachformat lautet:

```
FILE-CONTROL.
...
[ORGANIZATION IS] LINE SEQUENTIAL
...
```

Eine zeilensequenzielle 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 satzsequenziellen 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 zeilensequenziellen Datei mit einer aktuellen SAM-Datei erfolgt - wie bei satzsequenziellen Dateien - mittels ADD-FILE-LINK-Kommando (siehe Abschnitt „Zuweisen von katalogisierten Dateien").
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	S-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 zeilensequenzieller COBOL-Dateien in Bibliothekselementen ist das Vorhandensein der Bibliothek LMSLIB, die auf der TSOS-Kennung eingerichtet sein muss.
Beim Binden eines Programms, das zeilensequenzielle Dateien verarbeiten soll, muss zur Befriedigung von Externverweisen zusätzlich zu CRTE die Bibliothek $LMSLIB angegeben werden.

Beispiel 9-6

Erzeugen einer zeilensequenziellen Datei in einem Bibliothekselement

Einträge in der COBOL-Übersetzungseinheit:

```
...
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)'
```

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 [6] 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 COBOL2000 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 im Handbuch „COBOL2000-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 Druckertyp 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 stellen.	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 Druckersatzes erst ab dem zweiten Zeichen des Datensatzes.

Tabelle 20: 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 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 Druckervorschubzeichen, 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 vordefinierten Druckervorschubzeichen 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 21 und 22	Vorschubzeichen und -zeilen gemäß UNIX-Konventionen
PRINTER	wie oben	wie oben
PRINTER01-99	wie oben	nicht unterstützt
literal	wie oben	BS2000-Vorschubzeichen gem. Tabellen 21 und 22

In den folgenden Tabellen sind Druckervorschubzeichen 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 21: Druckervorschubzeichen

*) 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 22: Druckervorschubzeichen 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) in der COBOL-Übersetzungseinheit angesprochen werden können, gestattet es der SPECIAL-NAMES-Paragraf der ENVIRONMENT DIVISION, ihnen symbolische Namen zuzuordnen (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [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 ADD-FILE-LINK-Kommando erforderlich:

ADD-FILE-LINK dateiname, REC-FORM=*VAR(*ASA)

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 23: ASA-Vorschubsteuerzeichen und korrespondierende WRITE-Anweisungen

9.2.7 Verarbeiten von Dateien im ASCII- oder ISO-7-Bit-Code

Die Verarbeitung einer sequenziellen Datei im ASCII- bzw. ISO-7-Bit-Code unterstützt COBOL2000 durch die Klauseln (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [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-Paragrafen der CONFIGURATION SECTION und
- CODE-SET IS alphabetname-1 in der Dateierklärung der FILE SECTION.

ASCII-Code

Die erforderlichen Angaben in der COBOL-Übersetzungseinheit 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 in der Übersetzungseinheit Vereinbarungen analog denen für den ASCII-Code getroffen werden (siehe oben); es ist lediglich STANDARD-2 an Stelle des Schlüsselwortes STANDARD-1 in der ALPHABET-Klausel anzugeben. Bei Magnetbanddateien im ISO-7-Bit-Code gibt es darüberhinaus auch die Möglichkeit (siehe auch Abschnitt „Verarbeiten von Magnetbanddateien"), im ADD-FILE-LINK-Kommando für die Dateizuweisung SUPPORT=TAPE(CODE=ISO7) anzugeben.

9.2.8 Verarbeiten von Magnetbanddateien

Die Verarbeitung von Magnetbanddateien unterstützt COBOL2000 durch folgende Sprachmittel (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [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 Handbuch „COBOL2000-Sprachbeschreibung“ [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 ADD-FILE-LINK-Kommando zugewiesen und mit Attributen versehen werden (vgl. Abschnitt „Zuweisen von katalogisierten Dateien'' und Abschnitt „Festlegen von Dateimerkmalen''). 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)
/ADD-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 ADD-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 COBOL2000 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-Paragrafen der ENVIRONMENT DIVISION angegeben; ihr Format ist (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [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 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*sequenzielle Dateien steht nur der durch datenname-1 repräsentierte Ein-/ Ausgabezustand zur Verfügung.

Die Zustandswerte und ihre Bedeutung beziehen sich i.d.R. auf *satz*sequenzielle Dateien. Bei der Verarbeitung *zeilen*sequenzieller Dateien müssen bezüglich der Interpretation der Zustandswerte die spezifischen Eigenheiten der zeilensequenziellen Organisation (siehe Abschnitt „Zeilensequenzielle Dateien") berücksichtigt werden.

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 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 vor handene 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 zeilensequenzieller Verarbeitung: erfolgloser Zugriff auf PLAM-Element
34	Es wurde versucht, außerhalb der vom System festgelegten Bereichsgrenzen einer sequenziellen 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.
39	Die OPEN-Anweisung war aus einem der folgenden Gründe erfolglos: 1. Im ADD-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 ADD-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.
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 ADD-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder dem Format des verwendeten Datenträgers.

Tabelle 24: Ein-/Ausgabezustände für sequenzielle Dateien

9.3 Relative Dateiorganisation

In diesem Kapitel werden folgende Themen behandelt:

- Merkmale relativer Dateiorganisation
- COBOL-Sprachmittel für die Verarbeitung relativer Dateien
- Zulässige Satzformate und Zugriffsarten
- Eröffnungsarten und Verarbeitungsformen (relative Dateien)
- Erstellen einer relativen Datei mit wahlfreiem Zugriff
- Ein-/Ausgabezustände

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 sequenziellen 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 Handbuch [4]). Sie gestatten es mehreren Anwendern, gleichzeitig die Datei zu aktualisieren (siehe Abschnitt „Simultanverarbeitung von Dateien (SHARED-UPDATE)").
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 ADD-FILE-LINK-Kommandos der
FCBTYPE *UPAM (SAM wird mit Fehler abgewiesen) festgelegt wurde.

In folgenden Fällen ist nur die Angabe von FCBTYPE ISAM zulässig:

- bei expliziter Angabe einer variablen Satzlänge in der RECORD-Klausel und/oder
- bei Angabe von OPEN EXTEND und/oder
- bei Angabe von READ REVERSED

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 Abschnitt „Merkmale indizierter Dateiorganisation".

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 sequenzieller 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 X’FF’ als Kennzeichen für einen Leersatz (siehe Abschnitt „Eröffnungsarten und Verarbeitungsformen (relative Dateien)").
PAM-Dateien können nur auf „Key-Platten“ angelegt werden, d.h. im ADD-FILE-LINK Kommando ist die Angabe BLOCK-CONTROL=PAMKEY erforderlich (siehe Abschnitt „Platten- und Dateiformate").

9.3.2 COBOL-Sprachmittel für die Verarbeitung relativer Dateien

Das folgende Programmskelett gibt einen Überblick über die wichtigsten Klauseln und Anweisungen, die COBOL2000 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 in der Übersetzungseinheit 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 ADD-FILE-LINK-Kommando angefordert, im Stapelbetrieb die AT END-Bedingung ausgelöst,
- bei OPEN I-O oder OPEN EXTEND eine Datei mit dem Namen FILE.COBOL.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 Handbuch „COBOL2000-Sprachbeschreibung“ [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 Abschnitt „Eröffnungsarten und Verarbeitungsformen (relative Dateien)"):

SEQUENTIAL	legt fest, dass die Sätze nur sequenziell verarbeitet werden.
RANDOM	vereinbart, dass auf die Sätze nur wahlfrei zugegriffen wird.
DYNAMIC	gestattet, dass auf die Sätze wahlweise sequenziell 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 „Ein-/Ausgabezustände" 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 Abschnitt „Simultanverarbeitung von Dateien (SHARED-UPDATE)") 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 in der Übersetzungseinheit 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 ADD-FILE-LINK-Kommando verändert werden, wobei darauf zu achten ist, dass der Puffer mindestens so groß sein muss wie der längste Datensatz.

Außer bei neu angelegten Dateien (OPEN OUTPUT) hat die im Katalog eingetragene Blockgröße stets Vorrang gegenüber den Blockgrößenangaben im Programm bzw. im ADD-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 COBOL2000 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 wahlfreiem 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 Abschnitt „Eröffnungsarten und Verarbeitungsformen (relative Dateien)").

```
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 „Eröffnungsarten und Verarbeitungsformen (relative Dateien)" 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.
In der COBOL-Übersetzungseinheit 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... oder RECORD CONTAINS...TO...	(Format 2) (Format 3)

Tabelle 25: Satzformat und RECORD-Klausel

Zugriffsarten

Auf Sätze einer relativen Datei kann sequenziell, wahlfrei oder dynamisch zugegriffen werden.

In der COBOL- Übersetzungseinheit 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	Sequenzieller 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/vorhergehende Datensatz zur Verfügung gestellt. 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 sequenziell als auch wahlfrei zugegriffen werden. Die jeweilige Zugriffsart wird dabei über das Format der Ein-/Ausgabeanweisung gewählt.

Tabelle 26: ACCESS MODE-Klausel und Zugriffsart

9.3.4 Eröffnungsarten und Verarbeitungsformen (relative Dateien)

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 sequenziell 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.
- 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 sequenziell erfolgen.

- ACCESS MODE IS SEQUENTIAL

  erlaubt, eine relative Datei sequenziell 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.

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:

	Eintrag in der ACCESS MODE-Klausel

Anweisung	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 | PREVIOUS] [INTO...] [AT END...] [NOT AT END...] [END-READ...]	READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]	Für sequenziellen Zugriff: READ...{NEXT | PREVIOUS} [INTO...] [AT END...] [NOT AT END...] [END-READ] Für wahlfreien Zugriff: READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]

Tabelle 27: 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 sequenziell zu lesen. READ stellt dabei die Datensätze in der Reihenfolge aufsteigender (NEXT) und absteigender (PREVIOUS) relativer Satznummern zur Verfügung. 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 sequenzielle 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 sequenziell als auch wahlfrei zu lesen.
  Die jeweilige Zugriffsart wird dabei über das Format der READ-Anweisung gewählt (siehe Tabelle 27).
  Eine START-Anweisung ist nur für sequenzielles 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 | PREVIOUS] [INTO...] [AT END...] [NOT AT END...] [END-READ...]	READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]	Für sequenziellen Zugriff: READ...{NEXT | PREVIOUS} [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 28: 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 sequenziell zu lesen und dabei durch einen vorhergehenden START auf einen beliebigen Satz der Datei als Anfangspunkt zu positionieren. Darüber hinaus 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 sequenziell 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 RELATIVE-FILE
     ASSIGN TO "RELFILE"
     ORGANIZATION IS RELATIVE
    ACCESS MODE IS RANDOM ──────────────────────────────────────(1)
    RELATIVE KEY IS REL-KEY ────────────────────────────────────(2)
    FILE STATUS IS FS-CODE DMS-CODE. ──────────────────────────(3)
 DATA DIVISION.
 FILE SECTION.
 FD RELATIVE-DATEI.
 01 RELATIVE-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 DMS-CODE.
       06 DMS-CODE-1              PIC 9(2) COMP.
          88 DMS-CODE-2-DEFINED               VALUE 64.
       06 DMS-CODE-2              PIC X(4).
 01 CLOSE-SCWITCH                 PIC X      VALUE "0".
    88 DATEI-OFFEN                           VALUE "1".
    88 DATEI-GESCHLOSSEN                     VALUE "0".
 01 RELATIVE-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 RELATIVE-FILE.
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 DMS-CODE-2-DEFINED
         DISPLAY „DMS-CODE: „ DMS-CODE-2 UPON T
      END-IF
```

```
       IF DATEI-OFFEN
          CLOSE RELATIVE-DATEI
       END-IF
       DISPLAY „PROGRAMM ABNORMAL BEENDET" UPON T
       STOP RUN
   END-IF.
AUSGABE-FEHLER-ENDE.
        EXIT.
END DECLARATIVES.
VORLAUF.
    OPEN OUTPUT RELATIVE-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 RELATIVE-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 COBOL2000 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-Paragrafen der ENVIRONMENT DIVISION angegeben; ihr Format ist (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [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 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 data-name-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] ent-nommen werden.

	Bedeutung

Ein-/Ausgabe-Zustand	
	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 (sequenzielles 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 (sequenzielles 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 sequenziellen Zugriffsmodus zu geben, bei der die relative Satznummer so groß ist, dass sie nicht in das mit der RELATIVE KEY-Angabe beschriebene Datenfeld passt.
	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 auf Grund 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 ADD-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 ADD-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-Anweisung war 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.
46	Es wurde versucht, eine sequenzielle 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 sequenziellem 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 „Simultanverarbeitung von Dateien (SHARED-UPDATE)"): 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 „Simultanverarbeitung von Dateien (SHARED-UPDATE)"): die Aufruffolge READ - REWRITE/DELETE wurde nicht eingehalten.
95	Unverträglichkeit zwischen den Angaben im BLOCK-CONTROL-INFO- oder BUFFER-LENGTH-Operanden des ADD-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder dem Format des verwendeten Datenträgers
96	READ PREVIOUS wird nicht unterstützt für Module, die mit COBRUN ENABLE-UFS-ACCESS=YES kompiliert wurden, oder die Datei soll mit der DVS-Zugriffsart UPAM bearbeitet werden.

Tabelle 29: Ein-/Ausgabezustände für relative Dateien

9.4 Indizierte Dateiorganisation

In diesem Kapitel werden folgende Themen behandelt:

- Merkmale indizierter Dateiorganisation
- COBOL-Sprachmittel für die Verarbeitung indizierter Dateien
- Zulässige Satzformate und Zugriffsarten
- Eröffnungsarten und Verarbeitungsformen (indizierte Dateien)
- Positionieren mit START
- Ein-/Ausgabezustände

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 sequenziellen 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 Handbuch [4]). Sie gestattet es mehreren Anwendern, gleichzeitig eine Datei zu aktualisieren (siehe Abschnitt „Simultanverarbeitung von Dateien (SHARED-UPDATE)").

Indizierte Dateien können ausschließlich auf Plattenspeichern eingerichtet werden.

Dateistruktur

Eine ausführliche Beschreibung des Aufbaus einer ISAM-Datei findet sich in Handbuch [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 ADD-FILE-LINK-Kommando erreichen, dass der darin vereinbarte Prozentsatz eines Datenblockes 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 ADD-FILE-LINK-Kommando vereinbart:

```
/ADD-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 COBOL2000 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 in der Übersetzungseinheit 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 ADD-FILE-LINK-Kommando angefordert, im Stapelbetrieb dieAT END-Bedingung ausgelöst,
- bei OPEN I-O oder OPEN EXTEND eine Datei mit dem Namen FILE.COBOL.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 Handbuch „COBOL2000-Sprachbeschreibung“ [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 Abschnitt „Eröffnungsarten und Verarbeitungsformen (indizierte Dateien)"):

SEQUENTIAL	legt fest, dass die Sätze nur sequenziell verarbeitet werden.
RANDOM	vereinbart, dass auf die Sätze nur wahlfrei zugegriffen wird.
DYNAMIC	gestattet, dass auf die Sätze wahlweise sequenziell 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 sequenziellen 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 „Ein-/Ausgabezustände" 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 derBLOCK 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 Abschnitt „Simultanverarbeitung von Dateien (SHARED-UPDATE)") 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 in der Übersetzungseinheit 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 ADD-FILE-LINK-Kommando verändert werden (siehe Abschnitt „Festlegen von Dateimerkmalen"), wobei darauf zu achten ist, dass der Puffer mindestens so groß sein muss wie der längste Datensatz.

Außer bei neu angelegten Dateien (OPEN OUTPUT) hat die im Katalog eingetragene Blockgröße stets Vorrang gegenüber den Blockgrößenangaben im Programm bzw. im ADD-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 COBOL2000 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 COBOL2000-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 Abschnitt „Eröffnungsarten und Verarbeitungsformen (indizierte Dateien)").

```
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 „Eröffnungsarten und Verarbeitungsformen (indizierte Dateien)" 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.
In der COBOL-Übersetzungseinheit 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... oder RECORD CONTAINS...TO...	(Format 2) (Format 3)

Tabelle 30: Festlegen von Satzformaten in der RECORD-Klausel

Zugriffsarten

Auf Sätze einer indizierten Datei kann sequenziell, wahlfrei oder dynamisch zugegriffen werden.
In der COBOL-Übersetzungseinheit 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	Sequenzieller Zugriff: Die Datensätze können nur in der Reihenfolge ihrer Satzschlüssel verarbeitet werden. Das bedeutet: • Beim Lesen wird jeweils der nächste oder vorhergehende Datensatz zur Verfügung gestellt (für Primär- und Sekundärschlüssel). • Beim Schreiben wird jeweils der nächste Datensatz (mit aufsteigendem Primärschlüssel) 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 sequenziell als auch wahlfrei zugegriffen werden. Die jeweilige Zugriffsart wird dabei über das Format der Ein-/Aus gabeanweisung gewählt.

Tabelle 31: ACCESS MODE-Klausel und Zugriffsart

9.4.4 Eröffnungsarten und Verarbeitungsformen (indizierte Dateien)

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 sequenziell 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 | PREVIOUS] [INTO...] [AT END...] [NOT AT END...] [END-READ...]	READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]	Für sequenziellen Zugriff: READ...{NEXT | PREVIOUS} [INTO...] [AT END...] [NOT AT END...] [END-READ]
			Für wahlfreien Zugriff: READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]

Tabelle 32: 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 sequenziell zu lesen. READ stellt dabei die Datensätze in der Reihenfolge aufsteigender (NEXT) oder absteigender (PREVIOUS) 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 sequenzielle Leseoperationen fest (siehe auch Abschnitt „Positionieren mit START"). 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 sequenziell als auch wahlfrei zu lesen.
  Die jeweilige Zugriffsart wird dabei über das Format der READ-Anweisung gewählt (siehe Tabelle 33).
  Eine START-Anweisung ist nur für sequenzielles 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 | PREVIOUS] [INTO...] [AT END...] [NOT AT END...] [END-READ...]	READ... [INTO...] [KEY IS...] [INVALID KEY...] [NOT INVALID KEY...] [END-READ]	Für sequenziellen Zugriff: READ...{NEXT | PREVIOUS} [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 33: 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 sequenziell 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-Prozedur.

- ACCESS MODE IS DYNAMIC

  gestattet es, die Datei sowohl sequenziell 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 sequenzielle 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... und READ...PREVIOUS möglich ist, eine indizierte Datei sequenziell 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 PREVIOUS ──────────────────────────(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
         READ IND-DATEI PREVIOUS ————————————————————————————————(4)
             AT END
                SET VERARBEITUNGS-ENDE TO TRUE
             NOT AT END
                DISPLAY "NAECHSTER SATZSCHLUESSEL: " REC-KEY
                   UPON T
                PERFORM SATZ-VERARBEITEN
          END-READ
        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...PREVIOUS liest den Satz ein, auf den zuvor mit START positioniert wurde.

(4) READ...PREVIOUS liest 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 COBOL2000 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-Paragrafen der ENVIRONMENT DIVISION angegeben; ihr Format ist (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [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 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.

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 sequenziel lem 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 sequenzielles 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.
	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 ADD-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 Program-mangaben bzw. mit den entsprechenden Angaben im ADD-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.
46	Es wurde versucht, ein sequenzielles 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 sequenziellem 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 „Simultanverarbeitung von Dateien (SHARED-UPDATE)"): 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 „Simultanverarbeitung von Dateien (SHARED-UPDATE)"): 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 ADD-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder dem Format des verwendeten Datenträgers
96	READ PREVIOUS wird nicht unterstützt für Module, die mit COBRUN ENABLE-UFS-ACCESS=YES übersetzt wurden.

Tabelle 34: Ein-/Ausgabezustände für indizierte Dateien

9.5 Simultanverarbeitung von Dateien (SHARED-UPDATE)

In diesem Kapitel werden folgende Themen behandelt:

- ISAM-Dateien
- PAM-Dateien

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 ADD-FILE-LINK-Kommandos:

```
/ADD-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		
SHARED-UPDATE=YES		INPUT	I-O	OUPUT/ EXTEND	INPUT	I-O	OUTPUT/ EXTEND
	INPUT	X	X		X		
	I-O	X	X				
	OUTPUT/EXTEND						
SHARED-UPDATE=NO	INPUT	X			X		
	I-0						
	OUTPUT/EXTEND						

Tabelle 35: 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 COBOL2000-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 ADD-FILE-LINK-Kommando implizit oder explizit beim Erzeugen der Datei vereinbart wurde (siehe Abschnitt „Grundbegriffe zum Aufbau von Dateien").

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 ADD-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 ADD-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 auf Grund 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 ADD-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 ADD-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 auf Grund 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 9-12a).

   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 9-12b).
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 9-13) 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 Datensatzes 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 ADD-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 COBOL2000-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 Abschnitt „Indizierte Dateiorganisation").

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.

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 (auf Grund 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 Datei 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 so genannten „Deadlock“-Situationen kommen (siehe Beispiel 9-14).
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 Verarbeiten von XML-Dokumenten

In diesem Kapitel werden folgende Themen behandelt:

10.1 Bereitstellen von XML-Dokumenten

Abhängig von der Art der Verarbeitung kann ein COBOL-Programm ein XML-Dokument verarbeiten, das im Arbeitsspeicher oder in einer Datei bereitgestellt ist:

Verarbeitungsart	**XML-Dokument**	
	im Speicher	**in Datei**
strukturorientiert	X	X
ereignisorientiert	X	

Wenn das XML-Dokument in einer Datei bereitgestellt wird, spielt deren BS2000/OSD-Zugriffsmethode keine Rolle. Die Aufteilung des XML-Dokuments auf Dateisätze ist für das verarbeitende Programm sichtbar: An Stelle jeden Satzwechsels steht im XML-Dokument ein End of Line-Zeichen, d.h. das Äquivalent des ASCII-Zeichens X'0A' in demjenigen Zeichensatz, in dem das Programm Daten aus dem XML-Dokument erhält. Das Einschieben zusätzlicher End of Line-Zeichen kann jedoch auch unterdrückt werden, wenn beim Übersetzen die entsprechende Steuerung XML-LINE-FEED=IGNORED gesetzt ist, siehe Abschnitt „RUNTIME-OPTIONS-Option".

10.2 Verwenden von XML-Sprachmitteln in Programmen

Voraussetzung für die Übersetzung von Programmen, die Sprachmittel zur Verarbeitung von XML-Dokumenten nutzen, ist die Erkennung der mit den Sprachmitteln verbundenen neuen Schlüsselwörter. Dazu muss beim Übersetzen die entsprechende Steuerung XML-SUPPORT=YES gesetzt sein, siehe Abschnitt „SOURCE-PROPERTIES-Option".

Beim Ablauf derartiger Programme analysiert und zerlegt ein separates, **Parser** genanntes Open Source-Programmpaket die XML-Dokumente. Dieses Programmpaket ist nicht Bestandteil des COBOL-Compilers bzw. von CRTE, sondern steht im Internet zum Herunterladen zur Verfügung.

> ℹ Als Open Source-Software unterliegt die Nutzung dieses Programmpakets eigenen Lizenzbedingungen und Auflagen, die Sie beim Herunterladen aus dem Internet **akzeptieren** müssen.

Zum Ablaufzeitpunkt muss dieses Programmpaket als Modulbibliothek im BS2000/OSD zur Verfügung stehen. Wie Sie diese Modulbibliothek beschaffen, sofern sie noch nicht im BS2000/OSD bereitgestellt ist, wird in Abschnitt „Bereitstellen des Parsers" genauer erläutert.

10.3 Binden, Laden, Starten von Programmen mit XML- Sprachmitteln

Das Binden, Laden und Starten von Programmen, die Sprachmittel zur Verarbeitung von XML-Dokumenten verwenden, erfolgt im Prinzip so, wie in Kapitel „Binden, Laden, Starten" beschrieben.

Die Verarbeitung von XML-Dokumenten erfordert immer auch Zeichensatzkonvertierungen. Daher muss das Anschlussmodul GNLAPDT für entsprechende XHCS-Funktionen eingebunden werden:

Beim Binden mit TSOSLNK, siehe Abschnitt „Statisches Binden mit TSOSLNK", ist im Bindelauf dazu folgende, zusätzliche Anweisung nötig:

```
*RESOLVE ,$.SYSOML.XHCS-SYS.020    ———————————————————— (1)
```

(1) XHCS-Anschlussmodul: Es wird angenommen, dass die Bibliothek, die das Modul enthält, im System mit dem Namen SYSOML.XHCS-SYS.020 zur Verfügung steht.

Beim Binden mit dem BINDER, siehe Abschnitt „Binden mit dem BINDER", ist im Bindelauf dazu folgende, zusätzliche Anweisung nötig:

```
//RESOLVE-BY-AUTOLINK LIB=$.SYSOML.XHCS-SYS.020    ————————— (1)
```

(1) XHCS-Anschlussmodul: Es wird angenommen, dass die Bibliothek, die das Modul enthält, im System mit dem Namen SYSOML.XHCS-SYS.020 zur Verfügung steht.

Beim dynamischen Binden und Laden mit dem DBL, siehe Abschnitt „Dynamisches Binden und Laden mit dem DBL", ist dazu vor dem START PROGRAM- bzw. LOAD-PROGRAM-Kommando das folgende, zusätzliche Kommando nötig:

```
/ADD-FILE-LINK BLSLIB02,$.SYSOML.XHCS-SYS.020    ————————— (1)
```

(1) XHCS-Anschlussmodul: Es wird angenommen, dass die Bibliothek, die das Modul enthält, im System mit dem Namen SYSOML.XHCS-SYS.020 zur Verfügung steht.

Beim **Ablauf** eines COBOL-Programms mit XML-Anweisungen muss das ablaufende Programm auf die Modulbibliothek, die das Parser-Programmpaket enthält, zugreifen können, um daraus Module dynamisch nachladen zu können. Weisen Sie dazu die Bibliothek mit den Parsermodulen vor Ablauf des COBOL-Programms mittels des Linknamens COBPRSXM zu. Beim dynamischen Binden und Laden bzw. beim Laden von fertig gebundenen Programmen sind vor dem START PROGRAM- bzw. LOAD-PROGRAM-Kommando folgende zusätzliche Kommandos nötig:

```
/ADD-FILE-LINK COBPRSXM,$XYZ.SYSLIB.UTM-XML.030.RT    ——————— (1)
/ADD-FILE-LINK BLSLIB01,$.SYSLNK.CRTE          ———————————— (2)
```

(1) XML-Parser: Es wird angenommen, dass die Bibliothek, die die entsprechenden Module enthält, auf der Kennung XYZ mit dem Namen SYSLIB.UTM-XML.030.RT zur Verfügung steht.

(2) Die Module des Parsers benötigen das C-Laufzeitsystem. Weisen Sie daher das CRTE ebenfalls zu.

10.4 Zeichensatzerkennung

Für die korrekte Verarbeitung eines XML-Dokuments ist es entscheidend, den Zeichensatz zu kennen, der zur Darstellung des Dokuments verwendet wird. XML erlaubt die Angabe dieses Zeichensatzes in einer Zeichensatz-Deklaration innerhalb des Dokuments. Bei Datenübertragungen zwischen verschiedenen Datenverarbeitungssystemen erfolgen i.A. auch Konvertierungen der verwendeten Zeichensätze, jedoch keine inhaltlichen Änderungen. Das kann dazu führen, dass die Angabe des Zeichensatzes im XML-Dokument nicht mehr mit dem Zeichensatz übereinstimmt, der tatsächlich zur Darstellung verwendet wird.

Um die Zeichensatz-Deklaration im XML-Dokument erkennen zu können, muss bereits vorher für das Lesen des Dokuments eine Annahme über den verwendeten Zeichensatz getroffen worden sein. Dies ist näherungsweise möglich, da ein wohlgeformtes XML-Dokument immer mit der Zeichenfolge <?xml beginnen muss. Durch Vergleich des Dokumentanfangs mit der Darstellung dieser charakteristischen Zeichenfolge in den verschiedenen, vom Parser unterstützten Zeichensätzen, lässt sich ein aktuell für das XML-Dokument verwendeter Zeichensatz ableiten.

Darüber hinaus erlaubt das BS2000/OSD für Dateien die Vergabe eines Dateiattributs, das einen Zeichensatz benennt (CODED-CHARACTER-SET), erzwingt jedoch nicht, dass der Dateiinhalt in diesem Zeichensatz dargestellt ist. Bei der Bereitstellung von XML-Dokumenten im Arbeitsspeicher, die in COBOL zusätzlich möglich ist, lässt sich aus den Angaben im Programm ebenfalls ein Zeichensatz ableiten, der für die Darstellung des Dokuments verwendet wird, siehe Handbuch „COBOL 2000 Sprachbeschreibung“ [1], Abschnitt „ASSIGN-Klausel“.

Es gibt folglich drei Quellen, aus denen sich derselbe, zur Darstellung des Dokuments verwendete Zeichensatz ergeben sollte:

Z1 aus Untersuchung des Dokumentanfangs geschlossener, vermuteter Zeichensatz

Z2 externe Angabe des Zeichensatzes als Dateiattribut bzw. Angaben im Programm

Z3 Zeichensatz-Deklaration im XML-Dokument

Um die Notwendigkeit manueller Eingriffe vor der Verarbeitung eines XML-Dokuments weitestgehend zu vermeiden, akzeptiert das COBOL-System in gewissem Umfang auch fehlende bzw. widersprüchliche Angaben zu Zeichensätzen aus diesen drei Quellen.

Die Entscheidung für den letztlich zur Verarbeitung angenommenen Zeichensatz bzw. die Entscheidung für einen Ein-/Ausgabezustand bei nicht auflösbaren Widersprüchen erfolgt entsprechend der folgenden Tabelle. Ein Gedankenstrich (–) bedeutet, dass die Existenz bzw. Verträglichkeit für die Entscheidung keine Bedeutung hat.

Vorgefundene Situation						**Getroffene Entscheidung**	
Z1 erkannt *	**Z2 existiert ****	**Z3 existiert ****	**Z2 verträglich mit Z1 *****	**Z3 verträglich mit Z1 *****	**Z3 verträglich mit Z2 *****	**verwendeter Zeichensatz**	**Ein-/Ausgabe-Zustand**
ja	ja	ja	ja	-	ja	Z3	-
ja	ja	ja	ja	-	nein	Z2	-
ja	ja	ja	nein	ja	-	-	3D
ja	ja	ja	nein	nein	-	-	3D
ja	ja	nein	ja	-	-	Z2	-
ja	ja	nein	nein	-	-	-	3D
ja	nein	ja	-	ja	-	Z3	-
ja	nein	ja	-	nein	-	Z1	-

ja	nein	nein	-	-	-	Z1	-
nein	ja	-	-	-	-	Dokument in Datei: Z2	Dokument im Speicher: 3D
nein	nein	-	-	-	-	-	3D
-	unbekannter Zeichensatz	-	-	-	-	-	3D
-	-	unbekannter Zeichensatz	-	-	-	-	3D

* Als Zeichensatz Z1 kann nur UTF-16, EBCDIC oder UTF erkannt werden. Dabei steht EBCDIC als (unpräzise) Obermenge für alle speziellen Varianten (wie z.B. EDF03IRV, EDF041 usw.) und UTF als (unpräzise) Obermenge für UTF-8 und alle von XHCS unterstützten ISO-Varianten.

** Als Z2 für Dokumente in Dateien und als Z3 werden nur UTF-8, UTF-16, EBCDIC, ISO646, sowie die speziellen EBCDIC-Varianten bzw. ISO-Varianten unter dem Begriff 'existiert' verstanden, d.h. alle, die auch XHCS kennt. Alle anderen Zeichensätze werden als 'unbekannt' angesehen. Als Z2 für Dokumente im Speicher sind nur EBCDIC (für alphanumerische Datenfelder) und UTF-16 (für nationale Datenfelder) möglich.

*** ’Zeichensatz Zx verträglich mit Zeichensatz Zy' bedeutet, dass Zx und Zy den gleichen Zeichensatz bezeichnen, oder Zx ein genauer bezeichneter Zeichensatz aus der (unpräzisen) Obermenge Zy ist.

Wenn der letztlich ausgewählte Zeichensatz nur die unpräzise Obermenge EBCDIC bezeichnet, wird die zur Übersetzungszeit des Programms gültige spezielle Variante verwendet.

Wenn der letztlich ausgewählte Zeichensatz nur die unpräzise Obermenge UTF bezeichnet, wird UTF-8 verwendet.

Diese Zeichensatzerkennung erfolgt bei jeder OPEN DOCUMENT-Anweisung (ohne AT-Angabe) und während einer XML PARSE-Anweisung sowohl für das primäre XML-Dokument, als auch für die darin angesprochenen externen Entitäten bzw. DTDs.

10.5 Bereitstellen des Parsers

Um den XML-Parser aus dem Internet herunterzuladen und als Modulbibliothek bereitzustellen, gehen Sie folgendermaßen vor (siehe auch Handbuch „XML für openUTM“ [27]).

Die erforderliche Software wird zwar unter 'openUTM' im Internet bereitgestellt. Sie setzt jedoch **nicht** voraus, dass COBOL-Programme, die sie nutzen wollen, unter openUTM ablaufen müssen.

1. Gehen Sie auf die Internetseite *http://de.ts.fujitsu.com/openUTM* .
2. Folgen Sie den Anweisungen zum Download, akzeptieren Sie ggf. die Lizenzbedingungen und laden Sie die neueste Version der BS2000-Bibliothek auf Ihren PC.
3. Entpacken Sie das Element SYSLIB.UTM-XML.nnn.RT (nnn: Versionsangabe, mindestens 030) aus dem Archiv und übertragen Sie es auf den BS2000/OSD-Rechner: Dafür gibt es folgende zwei verschiedenen Vorgehensweisen:
   a. Entpacken Sie das Element aus dem Unterverzeichnis ftp und übertragen Sie es mit ftp binär.
   b. Entpacken Sie das Element aus dem Unterverzeichnis openft und übertragen Sie es mit openFT (Dateityp binär, Übertragungsmodus transparent).

Nun steht im BS2000/OSD eine PLAM-Bibliothek zur Verfügung: Dies ist die Modulbibliothek, die für den Ablauf der COBOL-Programme erforderlich ist.

10.6 Erweiterter Ein-/Ausgabe-Zustand für XML-Anweisungen (CBX-Code)

Die einfachen Ein-/Ausgabe-Zustände für XML-Dateien sind im Handbuch „COBOL 2000 Sprachbeschreibung“ [1] beschrieben.

Die kursiv dargestellten Fehler in der folgenden Tabelle treten nur bei der Zerlegung von DTDs auf.

Ein-/Ausgabe-Zustand	**Bedeutung**
0003	gleicher Name für Attribute mehrfach verwendet
0004	Kleinerzeichen ('<') im Attributwert
0005	öffnendes und schließendes Tag passen nicht zusammen
0010	doppelter Bindestrich in Kommentar
0011	Steueranweisung nicht beendet
0012	Name der Steueranweisung beginnt mit der reservierten Zeichenfolge 'xml' (Groß- bzw. Kleinschreibung spielt keine Rolle)
0013	ungültige hexadezimale Zeichenangabe in numerischer Zeichenentität
0014	ungültige dezimale Zeichenangabe in numerischer Zeichenentität
0016	numerische Zeichenentität stellt kein gültiges UTF-8-Zeichen dar
0017	ungültiges Zeichen in Entitätenreferenz Name
0102	leeres Dokument
0110	Wert eines Attributs nicht korrekt abgeschlossen
0119	CDATA-Abschnitt nicht korrekt abgeschlossen
0127	Gleichheitszeichen ('=') bei Attribut fehlt
0128	Wert für Attribut fehlt
0138	Zielname für Steueranweisung ungültig
0139	ungültiges Zeichen in Steueranweisung
0142	Versionsangabe in der XML-Deklaration fehlt
0148	nach dem Schlüsselwort ' encoding' fehlt das Gleichheitszeichen ('=')
0149	Zeichensatzname fehlt
0150	Zeichensatzname nicht korrekt abgeschlossen
0151	ungültiges Zeichen nach der Zeichensatz-Deklaration
0155	Wert für Standalone-Deklaration weder 'yes' noch 'no'
0160	Nach dem Ende des Dokuments folgt noch etwas Unerlaubtes
0315	Zeichensatz 'UTF-16LE' wird nicht unterstützt
0317	Zeichensatz nicht feststellbar

0320	EBCDIC-Zeichen in einem nationalen Datenfeld
0321	ASCII-/UTF-8-Zeichen in einem nationalen Datenfeld
1001	Parameter Entitätenreferenz am Dokumentende
1002	Parameter Entitätenreferenz im Prolog
1003	Parameter Entitätenreferenz im Epilog
1004	*Parameter Entitätenreferenz in Markup-Deklaration in interner DTD*
1005	Entität nicht deklariert
1006	Referenz auf nicht geparste Entität
1007	Referenz auf externe Entität in Attributwert
1008	unzulässige Referenz auf eine Parameter-Entität
1009	erwartete Zeichenfolge beginnt nicht mit Anführungszeichen
1010	Namensraum-Deklaration falsch
1012	*Wert in einer Entitäten-Deklaration ist fehlerhaft*
1013	*erwartetes Literal nicht gefunden*
1014	Leerzeichen fehlt
1015	erwarteter Name fehlt
1016	erwartetes Größerzeichen ('>') fehlt
1017	erwartetes Gleichheitszeichen ('=') fehlt
1018	Entität ist nicht ausbalanciert (Es wurde etwas begonnen, was nicht abgeschlossen ist oder abgeschlossen, was nicht begonnen wurde.)
1019	*nicht erlaubtes Zeichen ('&' oder '%') im Wert der Entitäten-Deklaration*
1020	*Parameter Entitätenreferenz im Wert einer Entität*
1021	*ungültiger URI im Wert einer Entität*
1022	*URI beginnt mit Nummernzeichen ('#')*
1023	Zeichensatz-Deklaration fehlt in der XML-Deklaration einer externen Entität
1024	externe DTD benötigt, obwohl durch Standalone-Deklaration ausgeschlossen
1025	externe Entität konnte nicht geladen werden
1098	Widerspruch zwischen externer Zeichensatzangabe und Zeichensatz-Deklaration
1099	Widerspruch zwischen internem Zeichensatz und Zeichensatz-Deklaration
2001	Zeichenfolge nicht korrekt abgeschlossen
2003	*Literal nicht korrekt abgeschlossen*
2004	Kommentar nicht korrekt abgeschlossen
2005	*Name in Notations-Deklaration fehlt*

2006	*Notations-Deklaration nicht korrekt abgeschlossen*
2007	*Fehler in Attributliste*
2008	*Attributliste nicht korrekt abgeschlossen*
2009	*Fehler in Element-Deklaration für gemischten Inhalt*
2010	*Element-Deklaration für gemischten Inhalt nicht korrekt abgeschlossen*
2011	*Fehler in Element-Deklaration*
2012	*Element-Deklaration nicht korrekt abgeschlossen*
2013	*Falsche oder fehlende XML-Deklaration*
2014	*XML-Deklaration nicht korrekt abgeschlossen*
2015	*Fehler in bedingtem Abschnitt*
2016	*bedingter Abschnitt nicht korrekt abgeschlossen*
2017	*ungültiger Inhalt in externer DTD*
2018	*Dokument Typ Definition nicht korrekt abgeschlossen*
2019	Zeichenfolge ’]]>’ für Ende CDATA-Abschnitt in einem Wert nicht erlaubt
2020	*Trennzeichen fehlt*
2021	*nmtoken in Attributliste fehlt*
2022	*Zeichenfolge '#PCDATA' fehlt in Element-Deklaration für gemischten Inhalt*
2023	*URI fehlt*
2024	*public identifier fehlt*
2026	*bedingter Abschnitt fehlerhaft*
2027	*Wert in Entitäten-Deklaration fehlt*
2028	hinter einer ausbalancierten Entität steht noch etwas
2029	Schachtelungstiefe von Entitätenreferenzen ist größer als 40
2030	*in einem bedingten Abschnitt fehlt das Schlüsselwort 'INCLUDE' oder 'IGNORE'*
2031	ungültiges Zeichen im Inhalt
2096	UTF-16-Zeichen in alfanumerischem Datenfeld
2097	ASCII-/UTF-8-Zeichen in alfanumerischem Datenfeld
2098	Widerspruch zwischen externer und internen Zeichensatzangabe
2099	Zeichensatz aus der Zeichensatz-Deklaration wird nicht unterstützt
2994	ILCS-Fehler
2995	unerwartetes EOF
2996	Fehler beim Lesen
2997	Vorausschau Puffer zu klein

2998	Speicher-Fehler
2999	interner Fehler / System-Fehler
3000	nicht verwendete Parser-Codes - bitte Systemverwalter verständigen

Tabelle 36: Erweiterte Ein-/Ausgabe-Zustände für XML-Anweisungen

11 Sortieren und Mischen

In diesem Kapitel werden folgende Themen behandelt:

11.1 COBOL-Sprachmittel zum Sortieren und Mischen

Das Sortieren und Mischen unterstützt COBOL2000 durch folgende Sprachmittel(siehe Handbuch „COBOL2000-Sprachbeschreibung“ [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 COBOL2000 die Sortierfunktion des BS2000 SORT (siehe Handbuch [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.

> **i** 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“
>   „ß“ mit „SS“ identifiziert
> - die Ziffern vor den Buchstaben einsortiert

11.2 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 ADD-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 ADD-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 Handbuch „COBOL2000-Sprachbeschreibung“ [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 Handbuch [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 Kapitel „Meldungen des COBOL2000-Systems").

Eingabedatei(en)

Ist keine Eingabeprozedur definiert, generiert COBOL2000 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 nn 99) dürfen nicht innerhalb eines Sortierprogramms verwendet werden.

Ausgabedatei

Ist keine Ausgabeprozedur definiert, generiert COBOL2000 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.

SORT-Parameterdateien

SORT erlaubt die Festlegung und Änderung von voreingestellten Werten für einige Parameter (siehe Anweisung MODIFY-SORT-DEFAULTS in [6]).

Die meisten dieser Werte wirken nicht bei SORT-Anweisungen in COBOL-Programmen. Daher werden solche Parameterdateien nur bei der ersten SORT-Anweisung in einer COBOL-Ablaufeinheit ausgewertet. Nachträgliche Änderungen bleiben für den weiteren Programmablauf wirkungslos. Dies führt zu einer Beschleunigung von COBOL-Programmen, die dynamisch sehr viele SORT-Anweisungen mit wenigen zu sortierenden Sätzen ausführen.

11.3 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 Kapitel „Fixpunktausgabe und Wiederanlauf") 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 Handbuch [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 E301; siehe Kapitel „Programmverknüpfungen"). 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 Kapitel „Fixpunktausgabe und Wiederanlauf" und Handbuch „Kommandos Band 1 - 6“ [3]).

11.4 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 Handbuch „COBOL2000-Sprachbeschreibung“ [1]).

11.5 Sortieren mit erweiterten Zeichensätzen

Beim Sortieren mit erweiterten Zeichensätzen wird das Format TRANSLATE-CHARACTER des SORT (siehe [6]) im BS2000/OSD genutzt.

Als Sprachmittel für das Sortieren mit erweiterten Zeichensätzen steht das Sonderregister SORT-CCSN (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]) bei der SORT-Anweisung (Datei- und Tabellensort) zur Verfügung[1].

Der Inhalt des Sonderregisters SORT-CCSN wird an den SORT übergeben als Name eines Moduls aus der Tabellenmodulbibliothek (SYSLNK.SORT.nnn.TAB[2]).

Diese Bibliothek enthält derzeit die Module EDF03DRV, EDF03IRV, EDF041. Um zusätzliche eigene Tabellen zu definieren, benötigt man die Berechtigung, diese Bibliothek zu ändern.

Zur Definition eigener Module stellt SORT in der Tabellenmodulbibliothek das Quellcodeelement MUSTER zur Verfügung (siehe auch „Hinweise für das Erstellen der TRANSLATE-CHARACTER-Tabellen“ in [6]).

Beispiel 11-1

Erzeugen von Dateien mit erweitertem Zeichensatz EDF041

Um mit einem Editor im BS2000 eine Datei im erweiterten Zeichensatz erstellen zu können, sind folgende Schritte nötig:

- Einstellungen der Emulation: Konfiguration … Datensichtstation
  DSS-Modus: 8 Bit
  Zeichensatz: Lat. Alphabet Nr. 9 ISO8859-15
  DSS-Typ: DSS9763

- Logische Eigenschaften der Datenstation ändern (siehe [3])

```
/MODIFY-TERMINAL-OPTIONS CODED-CHARACTER-SET=EDF041
```

- Einstellen des Codes im EDT für eine neue Datei (siehe [23]):

```
@CODENAM EDF04
```

Beispiel 11-2

Zuweisung einer Ausgabedatei mit erweitertem Zeichensatz:

```
/CREATE-FILE SORT-AUSGABE,CODED-CHARACTER-SET=EDF041 ——————————— (1)
/ADD-FILE-LINK LINK-NAME=AUSGABE,FILE-NAME=SORT-AUSGABE ———————— (2)
```

(1) weist das DVS an, die Datei SORT-AUSGABE mit dem CODED-CHARACTER-SET EDF041 anzulegen

(2) stellt die Beziehung zum Programm her

[1] Das Attribut CODED-CHARACTER-SET von SORT-Eingabe- oder Ausgabedateien wird vom COBOL-SORT nicht ausgewertet

[2] *nnn* steht für die aktuelle SORT-Version

12 Fixpunktausgabe und Wiederanlauf

Fixpunkte werden von COBOL2000-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 „COBOL2000 und POSIX").

12.1 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 Handbuch „COBOL2000-Sprachbeschreibung“ [1]):

```
RERUN [ON herstellername] EVERY {END OF {REEL | UNIT} | ganzzahl-1 RECORDS} OF
dateiname
```

herstellername

Angabe SYSnnn (0 <= nnn <= 244)
COBOL2000 erzeugt entweder eine Fixpunktdatei oder zwei Fixpunktdateien:

a. eine Fixpunktdatei, falls nnn <= 200.
   COBOL2000 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.
b. Zwei Fixpunktdateien, falls nnn > 200.
   COBOL2000 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 „Fixpunktausgabe für Sortierprogramme und Wiederanlauf" beschrieben.
Nach jeder fehlerfreien Ausgabe eines Fixpunktes werden dem Benutzer auf SYSOUT Informationen für einen eventuellen Wiederanlauf gemeldet.

Format der Meldung:

```
E301 CHECKPOINT#aa, 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

12.2 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 Handbuch „Kommandos Band 1 - 6“ [3]):

```
/RESTART-PROGRAM dateiname,REST-OPT=START-PROG(CHECKPOINT=NUMBER(...)
```

dateiname	Name der Fixpunktdatei (COBOL-Standardname; siehe Abschnitt „Fixpunktausgabe")
NUMBER(...)	Nummer der PAM-Seite, in der die Fixpunktsätze beginnen

i

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.

13 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.

13.1 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 bzw. die Unterprogrammadresse enthält. Abhängig von der Art des Unterprogrammaufrufs wird ein Programmsystem unterschiedlich gebunden und geladen.

Unterprogrammaufruf „CALL literal“ bzw. Programmadressbezeichner„ADDRESS OF PROGRAM 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“ bzw. „ADDRESS OF PROGRAM literal“, kann es, wie in Kapitel „Binden, Laden, Starten" beschrieben, zu einer permanent oder temporär ablauffähigen Programmausführungseinheit gebunden und anschließend geladen werden.

Unterprogrammaufruf „CALL bezeichner“ bzw. Programmadressbezeichner „ADDRESS OF PROGRAM 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 und Programmadressbezeichner „ADDRESS OF PROGRAM bezeichner“, 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, auf die es keine Externverweise (im Hauptprogramm) gibt, 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, auf die es keine Externverweise (im Hauptprogramm) gibt, 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, auf die es keine Externverweise (im Hauptprogramm) gibt, dynamisch nachladen.

Vor dem Aufruf des Bindeladers sollte folgende Zuweisung vorgenommen werden:

```
/ADD-FILE-LINK BLSLIBnn,laufzeitbibliothek
```

für das Common RunTime Environment (CRTE), das das COBOL-Laufzeitsystem enthält(hierfür darf die Bibliothek SYSLNK.CRTE.PARTIAL-BIND nicht verwendet werden, siehe CRTE-Benutzerhandbuch [2]).

Außerdem muss folgende Zuweisung vorgenommen werden:

```
/ADD-FILE-LINK COBOBJCT,bibliothek
```

für eine Bibliothek, die die nachzuladenden Unterprogramme enthält.

Dabei ist zu beachten, dass die Verwendung des Linknamens BLSLIBnn nur wirkt, wenn im Aufrufkommando für den DBL

```
RUN-MODE=ADVANCED(ALTERNATE-LIBRARIES=YES)
```

angegeben wird (siehe Abschnitt „Dynamisches Binden und Laden mit dem DBL").

Enthält die Ladeeinheit unbefriedigte WXTRNSs (dies ist z.B.dann der Fall, wenn im Unterprogramm weitere Dateien mit anderer Dateiorganisation als im Hauptprogramm verarbeitet werden) dann müssen die Operanden UNRESOLVED-EXTRNS=DELAY undLOAD-INFORMATION=REFERENCES angegeben werden:

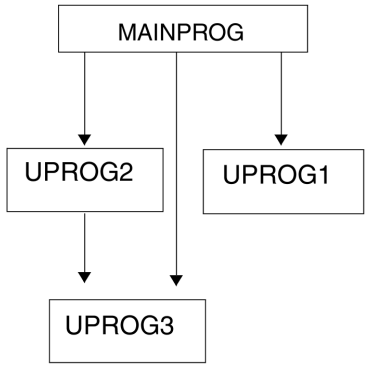
```
RUN-MODE=ADVANCED (ALTERNATE-LIBRARIES=YES, UNRES-EXT=DELAY, LOAD-INF=REF)
```

> Namen von Bindelademodulen (LLMs) können in CALL, CANCEL und ADDRESS OF PROGRAM als bezeichner angegeben werden, die bis zu 30 Zeichen lang sind. Für Objektmodule dürfen die Programmnamen nicht länger als acht Zeichen lang sein. Bei CANCEL bezeichner-Anweisungen für Programme, die im Objektmodulformat vorliegen, müssen diese Namen in der run unit in den ersten sieben Zeichen eindeutig sein und das achte Zeichen darf kein Bindestrich '-' sein.

Beispiel 13-1

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)

```
/ADD-FILE-LINK BLSLIB00,$.SYSLNK.CRTE
————————————————————————————(1)
/ADD-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.

2. Verwendung des TSOSLNK (Großmodulbinden)

```
/START-PROGRAM $TSOSLNK
MODULE GROSSMOD,LET=Y,UNSAT=N, LIB=MODUL.LIB
————————————————————————(1)
INCLUDE MAINPROG,BENUTZER-PROGRAMME
————————————————————————————————(2)
RESOLVE,BENUTZER-PROGRAMME
————————————————————————————————————————(3)
LINK-SYMBOLS *KEEP
————————————————————————————————————————————(4)
END

/ADD-FILE-LINK BLSLIB00,$.SYSLNK.CRTE
————————————————————————(5)
/ADD-FILE-LINK COBOBJCT,BENUTZER-PROGRAMME
————————————————————————(6)
/START-PROGRAM *MODULE(LIB=MODUL.LIB,ELEM=GROSSMOD,-
/RUN-MODE=ADVANCED(ALT-LIB=YES,UNRES-EXT=DELAY,-
————————————————(7)
/LOAD-INFO=REFERENCES))
```

(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) Mit dieser Anweisung werden die Symbole für die Einsprungstellen und die Programmabschnitte für den späteren Ablauf mit dem Bindelader sichtbar gehalten.

(5) Zuweisung der Laufzeitbibliothek

(6) Zuweisung der Bibliothek, aus der das Unterprogramm UPROG2 dynamisch nachgeladen wird

(7) Aufruf des Großmoduls GROSSMOD.

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

/ADD-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=REFERENCES))
```

(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.

b) Umwandeln der Objektmodule in einzelne Bindelademodule

```
/START-PROGRAM $BINDER
...
//START-LLM-CREATION INTERNAL-NAME=MAINPROG
(1)
//INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,
ELEM=MAINPROG                    |
//SAVE-LLM LIB=MODULE.LLM
(1)
...
//START-LLM-CREATION INTERNAL-NAME=UPROG1
(2)
```

```
//INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,
ELEM=UPROG1                    |
//SAVE-LLM LIB=MODULE.LLM,ENTRY-POINT=UPROG1
(2)
...
//START-LLM-CREATION INTERNAL-NAME=UNTER2
(3)
//INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,
ELEM=UNTER2                    |
//SAVE-LLM LIB=MODULE.LLM,ENTRY-POINT=UNTER2
(3)
...
//START-LLM-CREATION INTERNAL-NAME=UPROG3
(4)
//INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,
ELEM=UPROG3                    |
//SAVE-LLM LIB=MODULE.LLM
(4)
//END
...
/ADD-FILE-LINK BLSLIB00,$.SYSLNK.CRTE ————————————————————————————
(5)
/ADD-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-CREATI-ON-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.

13.2 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 die Prozedur zum nächsten SET-JOB-STEP, EXIT-JOB, LOGOFF, CANCEL-PROCEDURE, END-PROCEDURE und EXIT-PROCEDURE-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 (Achtung: Man kann den Funktionswert nicht mit der „RETURNING“- Angabe in der CALL-Anweisung erhalten).

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.

13.3 Parameterübergabe an fremdsprachige Programme

Mit COBOL-Prototypes können auch fremdsprachige Programme beschrieben werden. In diesem Fall stehen auch bei Aufruf fremdsprachiger Programme alle Möglichkeiten des erweiterten CALL Format 3 zur Verfügung (siehe Handbuch „COBOL2000-Sprachbeschreibung“ [1]). Andernfalls können nur die eingeschränkteren Möglichkeiten von Format1 und Format 2 genutzt werden.

Nähere Angaben zur Parameterübergabe sind im CRTE-Benutzerhandbuch [2] beschrieben.

13.4 Entladen von COBOL-Unterprogrammen

COBOL bietet kein Sprachmittel, um Unterprogramme zu entladen. Dafür muss der Anwender selber Assembler-Programme bereitstellen (siehe Abschnitt „UNBIND Makro“ im Handbuch „Bindelader-Starter“ [10]).

Bei solchen Entladevorgängen sind Abhängigkeiten der Module untereinander und zum COBOL-Laufzeitsystem im Rahmen des CRTE zu beachten:

- Solange noch andere COBOL-Module geladen bleiben, darf das COBOL Laufzeitsystem nicht entladen werden, z.B. wenn das zu entladende Modul als LLM oder Großmodul gebunden das Laufzeitsystem ganz oder teilweise enthält.
- Werden externe Dateien von COBOL-Programmen angesprochen, muss das COBOL-Laufzeitsystem auch dann geladen bleiben, wenn alle COBOL-Programme entladen wurden, aber zur weiteren Bearbeitung der externen Datei erneut COBOL-Unterprogramme nachgeladen werden sollen.
- Handelt es sich bei dem zu entladenden COBOL-Modul um eine Klassen- oder Interface-Definition, dann müssen alle Module, die diese Klassen- oder Interface-Definitionen verwenden oder davon erben, entladen werden.
- Wurde das zu entladende Modul mittels „CALL bezeichner“ von einem COBOL-Modul aus nachgeladen, das mit der Option OPTIMIZE-CALL-IDENTIFIER=YES übersetzt wurde, darf das entladene Modul nicht erneut mit dem „CALL bezeichner“ aufgerufen werden.

Beim Programmaustausch von Modulen unter openUTM sind die Anforderungen ebenfalls zu berücksichtigen.

Die Einhaltung der Regeln wird nicht vom COBOL-Compiler überprüft.

14 COBOL2000 und POSIX

In COBOL2000-BC nicht unterstützt !

Der COBOL Compiler 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:

- Handbuch „POSIX-Grundlagen“ [30]
- Handbuch „POSIX-Kommandos“ [29]

14.1 Überblick

Die folgenden drei Abschnitte bieten einen Überblick über den Einsatz des Compilers im POSIX-Subsystem.

- Übersetzen
- Binden
- Testen

14.1.1 Übersetzen

Für das Übersetzen von COBOL-Übersetzungseinheiten steht das POSIX-Kommando `cobol` zur Verfügung. Dieses Kommando ist in Abschnitt „Steuerung des Compilers" 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 Übersetzungsgruppen wird für jede Übersetzungseinheit ein LLM erzeugt, das in einer POSIX-Objektdatei abgelegt wird. *basisname* ist in diesem Fall für die zweite bis letzte Übersetzungseinheit der jeweilige ID-Name der Übersetzungseinheit, 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 "Allgemeine Optionen").

Erzeugen einer Übersetzungsliste

Mit der Option `-P` (siehe "Option zur Ausgabe von Übersetzungsprotokollen") können diverse Übersetzungslisten angefordert werden (z.B. Übersetzungseinheitliste, Fehlerliste, Querverweisliste). 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“ [29]).

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 EBCDIC-Code abgelegt Sofern von der Möglichkeit Gebrauch gemacht wird, das POSIX Dateisystem auf einem gemounteten UNIX-Dateisystem abzulegen bzw. mit Werkzeugen des UNIX -Betriebssystems 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.

Verwendung von Compilervariablen unter POSIX

Beim Aufruf des Compilers in POSIX im BS2000/OSD können die Werte der CompilerVariablen aus Environment-Variablen übernommen werden. In diesem Fall entfällt das Prefixing mit dem Namensteil „SYSDIR-“ (siehe auch Abschnitt „Zuweisung an Compilervariablen zur Steuerung der Quelltextmanipulation").

In POSIX haben Environment-Variable keinen Typ und ihr Inhalt wird im Programm als Zeichenkette bei der bedingten Compilation interpretiert.

14.1.2 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 "Allgemeine Optionen") 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`, "Optionen für den Bindelauf").

Die Module können von jedem ILCS-fähigen BS2000-Compiler erzeugte Module sein (z.B. COBOL85, COBOL2000, C, C++, ASSEMBH, FORTRAN90).

Wenn vom COBOL2000-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 COBOL2000-Laufzeitsystem werden vom Binder automatisch aus der CRTE-Bibliothek $.SYSLNK.CRTE.PARTIAL-BIND aufgelöst.

Binde-Reihenfolge

1. Alle vom Compiler bei der Übersetzung generierten Objektdateien mit INCLUDE-MODULES-Anweisungen.
2. 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.
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 eigeneRESOLVE-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 1.) richtet sich die Bindereihenfolge nach der Reihenfolge der zugehörigen Quelldateien.

Beispiel 14-1

```
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. cobupro1.o
2. cobupro2.o
3. cobbsp.o
4. cobupro3-5.a
5. /usr/private/libxyz.a
6. $MYTEST.LIB1
7. $MYTEST.LIB2
8. Laufzeitbibliotheken

14.1.3 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 "Testhilfe-Option") 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 TesthilfeKommandos so eingegeben werden, wie im Handbuch „AID“ [8] beschrieben. Nach Beendigung des Programms ist wieder die POSIX-Shell die aktuelle Umgebung.

Das `debug`-Kommando ist im Handbuch „POSIX-Kommandos“ [29] beschrieben.

14.2 Bereitstellen der Übersetzungseinheit

Der COBOL2000-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 "Allgemeine Optionen").

Übersetzungseinheiten, 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 zur POSIX-Shell mit rlogin, steht zum Editieren das POSIX-Kommando `vi` zur Verfügung (siehe Handbuch „POSIX-Kommandos“ [29]).

Eingabe von Programmteilen (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 14-2

COPY-Anweisungen in der Übersetzungseinheit:

```
...
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.

14.3 Steuerung des Compilers

Der COBOL2000-Compiler kann in der POSIX-Shell mit dem Kommando **cobol** aufgerufen und mit Optionen gesteuert werden.

Aufruf-Syntax

`cobol` *'BLANK'option'BLANK'* ... *eingabedatei'BLANK'* ...

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, -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.
5. Dem Compiler unbekannte Optionen, dh. Optionen, die nach dem Bindestrich („-“) mit einem unbekannten Buchstaben beginnen, werden an den Binder cobld weitergereicht. Steht zwischen der unbekannten Option und einem Argument ein Leerzeichen, so wird diese als Option ohne Argument interpetiert und weitergereicht.

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 mit dem Binder cobld gestartet.

Die Optionen zur Steuerung des Übersetzungs- und Bindelaufs sind nachfolgend beschrieben.

14.3.1 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 eine Übersetzungseinheit 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 Treibers des COBOL2000-Compilers und des cobol-Kommandos
- Meldungen des COBOL2000-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 COBOL2000-Compilers.

–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

14.3.2 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 „Steuerung des Compilers mit COMOPT-Anweisungen" beschrieben.

Beispiel 14-3

```
-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 Abkürzungen**
ACCEPT-LOW-TO-UP={YES/NO}	ACC-L-T-U
ACTIVATE-WARNING-MECHANISM={YES/NO}	ACT-W-MECH
ACTIVATE-XPG4-RETURNCODE={YES/NO}	
ALIGN-LLM-PAGE={YES/NO}	A-L-P
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
DEFAULT-CALL-CONVENTION={COBOL/COMPATIBLE}	DEF-C-C
ENABLE-COBOL85-KEYWORDS-ONLY={YES/NO}	
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
IGNORE-OPTION-DIRECTIVES={YES/NO}	IGN-O-DIR
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=1..100	MAX-ERR
MERGE-DIAGNOSTICS={YES/NO}	M-DIAG
MERGE-REFERENCES={YES/NO}	M-REF
PERMIT-STANDARD-DEVIATION={YES/NO}	P-S-D
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}	
SUPPRESS-LISTINGS={YES/NO}	SUP-LIST
SUPPRESS-MODULE={YES/NO}	SUP-MOD
TERMINATE-AFTER-SEMANTIC={YES/NO}	TERM-A-SEM
TERMINATE-AFTER-SYNTAX={YES/NO}	TERM-A-SYN
TEST-WITH-COLUMN1={YES/NO}	TEST-W-C
UPDATE-REPOSITORY={YES/NO}	UPD-R
USE-APOSTROPHE={YES/NO}	USE-AP

14.3.3 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
OBJECT
NOOBJECT
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 14-4

```
-P "(ALL,NOXREF)"
```

14.3.4 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 „Binden".

–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.
Die -L-Optionen müssen vor den -l-Optionen angegeben werden, für die sie gelten sollen.

–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. Man beachte dabei: für die Ausgabedatei sind nicht nur Schreib-, sondern auch Leserechte erforderlich.

–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. Gleiches gilt für die Laufzeitsystembibliothek lib*c*.a.
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 ShellUmgebungsvariablen BLSLIB*nn* (00 *nn* 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 14-5

```
export BLSLIB00='$RZ99.SYSLNK.COB.999'
export BLSLIB01='$MYTEST.LIB'
cobol mytest.o -l BLSLIB -M MYTEST
```

14.3.5 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.

14.3.6 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`

Die Dateien mit dem Suffix .cob oder .cbl sind die Eingabequellen für den COBOL2000-Compiler. Der COBOL2000-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 "Allgemeine Optionen ").

Die Dateien mit dem Suffix .o und .a sind die Eingabequellen für den Binder.

Die Dateinamen mit anderen Suffixen werden an den Binder cobld weitergereicht.

14.3.7 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 "Optionen für den Bindelauf") 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 Übersetzungsgruppen werden die Namen der LLM-Objekt-dateien für die zweite bis letzte Übersetzungseinheit aus dem ID-Namen der Übersetzungseinheit und dem Suffix .o gebildet (siehe auch Abschnitt „Übersetzen").

14.4 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 Übersetzungseinheiten je eine Übersetzungseinheit- 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 Modul `upro.o` zu einer ausführbaren Datei `hugo`.
Das Programm mit dem PROGRAM-ID-Namen BSPPROG wird zum Hauptprogramm.

14.5 Unterschiede zu COBOL2000 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.

14.5.1 Sprachfunktionale Einschränkungen

Die im Folgenden aufgeführten Sprachmittel des COBOL2000-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 und kann zum Abbruch des Programmlaufes führen.

Programmadressbezeichner

„ADDRESS OF PROGRAM bezeichner“ erfordert wie „CALL bezeichner ...“ dynamisches Nachladen zum Ablaufzeitpunkt und ist deshalb im POSIX nicht möglich.

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 (SHARED-UPDATE) ist in POSIX nicht möglich.
- Im POSIX wird das im UNIX-Betriebssystem übliche LOCKING-Verfahren umgesetzt. So wird z.B. das mehrfache Öffnen der gleichen Datei zur Ausgabe nicht unterbunden.
- In der ALPHABET-Klausel spezifizierter Zeichensatz STANDARD-2 (International Reference Version of the ISO 7-Bit Code) wird in der CODE-SET Klausel nicht unterstützt. Ein derartiger OPEN wird zur Laufzeit mit FILE STATUS 30 abgewiesen.
- In den Meldungen COB9151 und COB9175 bei Fehlern beim POSIX-Dateizugriff wirdstatt DMS-Codes die entsprechenden SIS-Meldungsnummern eingesetzt. Das gleiche gilt auch für den an das COBOL-Objekt zurückgegebenen „extended“ File Status. Auch der zurückgegebene File Status kann vom bisher erwarteten Wert abweichen (siehe Abschnitt „Ein-/Ausgabezustände").
- READ PREVIOUS wird nicht unterstützt und mit File Status 96 abgewiesen.

XML-Dokumente

Die neuen Sprachmittel zum Lesen von XML-Dokumenten erfordern dynamisches Nachladen zum Ablaufzeitpunkt des Programms und sind deshalb in POSIX nicht möglich.

14.5.2 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 Handbuch „COBOL2000-Sprachbeschreibung“ [1]).

Beispiel 14-6

```
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
cmdargument-1<AAAA >
cmdargument-2<BBB  >
cmdargument-3<CC   >
cmdargument-4<D    >
argument-2:BBB  :
```

14.5.3 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 COBOL2000-Herstellernamen in ACCEPT-/DISPLAY-Anweisungen zur Ein-/Ausgabe kleiner Datenmengen sind in POSIX folgende Standard-Ein-/Ausgabeströme zugeordnet:

COBOL2000	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 „Programmablauf in der POSIX-Shell".
- 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.
- COB90xx-Meldungen gehen im POSIX auf stderr.

Repository-Nutzung

Zuweisen eines oder mehrerer Repositories für die Eingabe und eines für die Ausgabe ist nicht möglich. Es steht nur das Default-Repository SYS.PROG.LIB im BS2000 für diesen Zweck zur Verfügung (eine Zuweisung ist nicht nötig).

14.6 Verarbeiten von POSIX-Dateien

In diesem Kapitel werden folgende Themen behandelt:

- Programmablauf in BS2000-Umgebung
- Programmablauf in der POSIX-Shell
- Ein-/Ausgabezustände

14.6.1 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.
- keine Unterstützung von Pseudo-Dateien (siehe ADD-FILE-LINK im BS2000/OSD Benutzerhandbuch „Kommandos“ [3].
- 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.
- In der ALPHABET-Klausel spezifizierter Zeichensatz STANDARD-2 (International Reference Version of the ISO 7-Bit Code) wird in der CODE-SET Klausel nicht unterstützt. Ein derartiger OPEN wird zur Laufzeit mit FILE STATUS 30 abgewiesen.
- In den Meldungen COB9151 und COB9175 bei Fehlern beim POSIX-Dateizugriff wird statt DMS-Codes die entsprechenden SIS-Meldungsnummern eingesetzt. Das gleiche gilt auch für den an das COBOL-Objekt zurückgegebenen „extended“ File Status. Auch der zurückgegebene File Status kann vom bisher erwarteten Wert abweichen (siehe Abschnitt „Ein-/Ausgabezustände").
- Die Nutzung von Dummy-Dateien wird nicht unterstützt.
- Dateien >32 Gbyte können verarbeitet werden, ohne dies im /ADD-FILE-LINK-Kommando extra einschalten zu müssen.

Zuweisen einer POSIX-Datei

Die Zuweisung einer POSIX-Datei erfolgt mit einer S-Variablen namens SYSIOL-externer-name, wobei SYSIOL- ein fester Namensbestandteil ist undexterner-name den Linknamen aus der ASSIGN-Klausel des Programms enthalten muss. externer-name darf keine Kleinbuchstaben enthalten.

Die S-Variable wird mit dem Kommando SET-VARIABLE folgendermaßen initialisiert:

```
/[SET-VAR] SYSIOL-externer-name= {  '*POSIX(dateiname)'
                                 | '*POSIX(relative-pfadname)'
                                 | '*POSIX(absolute-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 14-7

für gemischte Dateiverarbeitung

COBOL-Übersetzungseinheit:

```
...
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:

```
/ADD-FILE-LINK CUST2,CUST.FILE
```

14.6.2 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 "Sprachfunktionale Einschränkungen").

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 initialisiert und mit dem POSIX-Kommando export exportiert werden.
Die Umgebungsvariable wird folgendermaßen initialisiert:

```
external-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 14-8

COBOL-Übersetzungseinheit:

```
...
FILE-CONTROL.
SELECT AFILE ASSIGN TO "CUST1"
...
```

Verknüpfung mit der POSIX-Datei cust1 vor Aufruf des Programms:

```
export CUST1=/USERIDXY/customers/cust1
```

14.6.3 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 COBOL2000 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-Paragrafen der ENVIRONMENT DIVISION angegeben; ihr Format ist z.B. in Abschnitt „Ein-/Ausgabezustände" dargestellt.
Die beiden in der FILE STATUS-Klausel defibrinieren 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 SIS-Codes (POSIX) 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 sequenziellem 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 CREATE-FILE oder MODIFY-FILEATTRIBUTES-Kommando
35	SRI	OPEN INPUT/I-O auf nicht vorhandene Datei
38	SRI	OPEN auf eine mit CLOSE WITH LOCK geschlossene Datei
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
47	S	READ auf nicht zum Lesen geöffnete Datei
	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
96	R1	READ PREVIOUS nicht unterstützt

*) S = sequenzielle Organisation, R = relative Organisation, I = indexsequenzielle 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 sequenziell wurde kein read sequenziell 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
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
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

15 Nutzbare Software für COBOL-Anwender

In diesem Kapitel werden folgende Themen behandelt:

- Advanced Interactive Debugger AID
- Library Maintenance System LMS
- Jobvariablen
- Datenbankschnittstelle ESQL-COBOL
- Universeller Transaktionsmonitor openUTM
- Entwicklungsumgebung Net Express® mit BS2000/OSD-Option

15.1 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 einer COBOL-Übersetzungseinheit zur Adressierung verwendet werden. Das nichtsymbolische 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 Übersetzungseinheit-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 Verwendung 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- bzw. INVOKE- 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 COBOL2000 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 (Paragrafen, 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 [20]
AID Testen von COBOL-Programmen [8]
AID Testen auf Maschinencodeebene [21].

15.2 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 eine Übersetzungseinheit. 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 [11]

15.3 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]

15.4 Datenbankschnittstelle ESQL-COBOL

Charakterisierung des Produktes

ESQL-COBOL (BS2000/OSD) V3.0 realisiert für COBOL-Anwendungen im BS2000/OSD die Anwender-Programmschnittstelle „embedded SQL“ zum Datenbanksystem SESAM/SQL-Server V5.0.

Der SQL-Funktionsumfang von SESAM/SQL-Server V5.0 kann mit ESQL-COBOL (BS2000/OSD) V3.0 uneingeschränkt genutzt werden.

ESQL-COBOL (BS2000/OSD) V3.0 ist als SQL-Precompiler lediglich zur Programmentwicklung erforderlich. Das SQL-Laufzeitsystem ist Bestandteil von SESAM/SQL-Server.

Für den Einsatz von ESQL-COBOL (BS2000/OSD) V3.0 ist stets das SQL-Laufzeitsystem erforderlich. Für die Vorübersetzung von embedded SQL-COBOL-Programmen wird somit der SESAM/SQL Server V5.0 benötigt, auch wenn die Vorübersetzung ohne Prüfung gegen das Datenbank-Schemata erfolgt.

Umfang der SQL-Funktionen

- Suchen von Datensätzen (SELECT-Anweisung) einschließlich höherer Funktionen wie Join, Arithmetik, Aggregatsfunktionen (z.B. Durchschnittsbildung),
- Neuaufnahme, Ändern, Löschen von Datensätzen.

Mit SESAM/SQL lassen sich Daten manipulieren und Funktionen zur Administration von Datenbanken ausführen (siehe Handbuch „SESAM/SQL-Server (BS2000/OSD)“ [17].

Technische Hinweise

Die SQL-Anweisungen eines ESQL-COBOL-Programms sind in den COBOL-Code eingebettet und werden von einem Precompiler durch COBOL Quellcode ersetzt. Die Ausgabe des Precompilers ist eine normale COBOL-Quelle, die mit dem COBOL2000-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 [16] - [18]

15.5 Universeller Transaktionsmonitor openUTM

openUTM 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 Formatierungssystem FHS wird die Ein-/Ausgabe über Formate für Datenstationen unterstützt.

openUTM garantiert, dass eine Transaktion mit allen Datenänderungen entweder vollständig oder nicht durchgeführt wird, und gewährleistet die Konsistenz der Anwenderdaten in Kombination mit UDS/SQL, SESAM /SQL, LEASY und PRISMA.

openUTM bietet Wiederanlauffunktionen bei Anwendungsabbruch, Netzausfall/Netzstörung oder Bildschirmstörungen. openUTM 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 openUTM-Anwendungen arbeiten.

Für Abrechnungszwecke steht ein auf den Teilhaberbetrieb abgestimmtes Accounting-Verfahren zur Verfügung.

openUTM bietet umfangreiche Datenschutzmechanismen für den Zugang zu Anwendungen und die Auswahl von Teilfunktionen einer Anwendung.

openUTM dient als Basis für eine Reihe weiterer Softwareprodukte.

Dokumentation

openUTM-Handbücher [23] - [26]

15.6 Entwicklungsumgebung Net Express® mit BS2000/OSD-Option

Der langjährige Partner Micro Focus bietet mit Net Express und der BS2000/OSD-Option eine auf Windows-Systemen ablaufende Entwicklungsumgebung für die Entwicklung von BS2000-COBOL-Anwendungen an.

Die Entwicklungsumgebung von Net Express bietet mit der BS2000/OSD-Option alle Funktionen für eine schnelle und effiziente Entwicklung von BS2000-Anwendungen. Neben reinen Batch- und openUTM-Anwendungen mit Zugriffen auf die Datenhaltungssysteme LEASY, SESAM/SQL und UDS/SQL können auch Client/Server-Applikationen entwickelt und auf dem PC getestet werden.

Batch- und Dialog-Anwendungen, die den Transaktionsmonitor openUTM nutzen, können mit Net Express auf der Windows-Workstation entwickelt und getestet werden, bevor sie auf einer BS2000-Plattform in den produktiven Einsatz gebracht werden. Die Verlagerung der Entwicklungsaktivitäten auf den PC bringt entscheidende Verbesserungen bei der Produktivität und der Software-Qualität.

Client/Server-Anwendungen, die einen BS2000/OSD-Server nutzen, können mit der BS2000/OSD-Option unter Net Express entwickelt und getestet werden. Net Express stellt eine einheitliche Entwicklungsumgebung sowohl für den Client- als auch für den Server-Teil der Anwendung bereit, liefert für den produktiven Einsatz auf dem Client die Laufzeitumgebung und ermöglicht mit der BS2000/OSD-Option den gemeinsamen Test von Client und Server auf einer Plattform.

Integrierte Entwicklungsumgebung

Net Express verfügt über eine hocheffiziente Entwicklungsumgebung, die mit der BS2000/OSD-Option zu einer kompletten Suite mit Tools und Assistenten zur Anwendungsentwicklung erweitert wird. Schon bei der Projekterstellung wird der Anwender von einem Assistenten geleitet, um BS2000-spezifische Optionen automatisch generieren zu können.

Mit Hilfe der Projektverwaltung können auf einfache Weise auch sehr umfangreiche Applikationen gepflegt werden. Die einzelnen Generierungsschritte werden einmal definiert und können dann über die Rebuild-Funktion per Mausklick gestartet werden. Der auf COBOL-Programmierer zugeschnittene Editor vereinfacht außerdem das Ändern von Quellcode. Die Kontrollfunktionen zur Verwaltung der Sourcen ermöglichen die Zusammenarbeit in Teams, ohne dass die Programmierer ihre Änderungen gegenseitig überschreiben.

Moderner COBOL-Compiler

Net Express enthält einen modernen Compiler, der auf den bewährten Stärken von COBOL basiert und für die Entwicklung von BS2000-Anwendungen die folgenden Highlights bietet:

- Kompatibilität zum COBOL2000-Compiler des BS2000 kann über eine Direktive eingestellt werden. Alle Konstrukte, die diesem Sprachumfang nicht genügen, werden als inkompatibel angezeigt.
- Volle Unterstützung des BS2000-EBCDIC-Codes mit der BS2000/OSD-Option.
- Unterstützung für objektorientierte COBOL Entwicklung und Debugging.
- Funktionen zur Simulation der BS2000-Systemumgebung:

  Für Entwicklungszwecke auf dem PC bildet die Net Express BS2000/OSD-Option bestimmte Funktionen der Laufzeitumgebung des BS2000 nach. Dazu gehören User- und Task-Switches, deren Einstellungen in Dateien hinterlegt werden können, damit sie den Applikationen zur Laufzeit zur Verfügung stehen. Ein Tool setzt Job-Variablen in der Entwicklungsumgebung von Net Express und stellt sie für die Anwendung bereit.

openUTM-Simulation

Die openUTM-Simulation der BS2000/OSD-Option basiert auf den Definitionen für dieKDCDEF-Utility im BS2000. Damit wird eine openUTM-Anwendung beschrieben mit all ihren Parametern wie Transaktionen und Teilprogrammen. Mit dem Tool KDCCECK können auf dem PC entworfene oder erweitete KDCDEF-Dateien auf ihre syntaktische Korrektheit überprüft werden. Der BS2000-Offloading-Wizard setzt alle für die Kompilierung und den Bindelauf einer openUTM-Anwendung erforderlichen Parameter und sorgt beim Testen

automatisch für den richtigen Start der Anwendung. Zur Kommunikation zwischen den Anwendungsprogrammen und openUTM wird die KDCS-Schnittstelle unterstützt. Deren Parameter können zur Laufzeit der Anwendung mit der TRACE Funktion angezeigt und interaktiv verändert werden.

Damit werden die Anwendungslogik und die dazugehörenden Transaktionsklammern, die Verknüpfung der einzelnen Teilprogramme über openUTM und die Steuerung der Maskenausgabe visualisiert und eine schnelle Fehlerlokalisierung ermöglicht. Zur Unterstützung formatierter Dialoge wird das Formatierungssystem FHS nachgebildet. IFG-Formatbibliotheken können auf dem Host entladen und zum PC übertragen werden.

Dort können sie mit einem speziellen Maskeneditor (SMSEDX) bearbeitet und anschließend zum Host übertragen werden. Zur Laufzeit der Anwendung auf dem PC wird auf die Maskenbibliothek zugegriffen. Die entsprechende Maske wird nach den Regeln des BS2000-Formatierungssystems FHS formatiert, wobei Farbeinstellungen konfiguriert werden können.

Mit dem Dialog-Test-Recorder können die Tastatureingaben und Bildschirmausgaben formatierter openUTM-Dialoge protokolliert werden. Diese Aufzeichnungen lassen sich zur automatischen Wiederholung eines einmal protokollierten Testlaufs verwenden. Ein spezieller Viewer (DTRVIEW) ermöglicht das Vergleichen der protokollierten Testergebnisse und erlaubt bei Unterschieden eine schnelle Fehleranalyse.

Simulation des openUTM-Client

Zur Entwicklung von Client/Server-Applikationen mit einer openUTM-Anwendung als Server enthält die Net Express BS2000/OSD-Option eine Simulation des openUTM-Client. Damit kann auch der Client mit Net Express entwickelt und getestet werden. Beim Test kann die Client-Applikation mit einer Server-Anwendung kommunizieren, die unter der openUTM-Simulation der BS2000/OSD-Option läuft. So kann das Zusammenspiel beider Anwendungsteile auf einer Plattform ausgetestet werden. Dabei lassen sich beide Anwendungsteile gleichzeitig animieren, und der Server kann mit dem openUTM-TRACE überwacht werden. Für die Kommunikation mit einer “echten” openUTM-Anwendung ist die entsprechende Software erforderlich.

Simulation der Datenhaltungssysteme LEASY, SESAM/SQL und UDS/SQL

Die Net Express BS2000/OSD-Option ermöglicht bei der Installation die Selektion von Simulationsmodulen für die BS2000-Datenhaltungssysteme LEASY, SESAM/SQL und UDS/SQL. Alle DB-Simulationen beinhalten verschiedene Dienstprogramme, die bei der Installation in die integrierte Entwicklungsumgebung von Net Express aufgenommen werden. Sie ermöglichen die Übernahme von Strukturinformationen und Testdatenbeständen aus Datenbanken des Hosts. Mit den DB-Simulationsmodulen können BS2000-Anwendungen, die diese Datenbanksysteme nutzen, in vollem Umfang gewartet und weiterentwickelt oder neu implementiert werden, unabhängig davon, ob es sich um Batch- oder Dialogan-wendungen handelt. Bereits beim Anlegen eines solchen Projektes werden vom BS2000-Offloading-Assistenten die entsprechenden Datenbank-spezifischen Einstellungen für den Compiler und den Binder generiert.

Die Datenbank-Simulationsmodule führen die Datenbankzugriffe auf dem PC durch und verhalten sich an der Schnittstelle zum COBOL-Programm völlig analog zur OriginalDatenbank auf dem BS2000. Das gilt insbesondere für die zurückgegebenen Parameter wie z.B. den Datenbankstatus, so dass auch Fehlersituationen in der Simulation ausgetestet werden können. Innerhalb von openUTM-Anwendungen werden auch in der Simulation die Transaktionen der Datenbanken mit openUTM-Transaktionen koordiniert. Dadurch kann auch dieses Zusammenspiel mit der BS2000/OSD-Option getestet werden.

TRACE-Funktion für Datenbankzugriffe

Alle Datenbanksimulationen sind mit einer komfortablen TRACE-Funktion mit grafischer Oberfläche ausgestattet. An der Schnittstelle zwischen COBOL-Programm und Datenbank können alle übergebenen Parameter angezeigt und interaktiv verändert werden. Die Anzeige ist sowohl vor als auch nach dem Aufruf möglich, sodass die Auswirkungen der jeweiligen Aktion sofort analysiert werden können. Hilfefunktionen erläutern mögliche Ursachen für Datenbank- oder Zugriffsfehler.

Die TRACES bieten die Möglichkeit, zwischen unterschiedlichen Darstellungsformen der Datenbankzugriffe umzuschalten. Während für den einzelnen Zugriff auf die Datenbank die aktuellen Parameterinhalte der CALL-

Schnittstelle angezeigt werden können, lässt sich jederzeit die aktuelle Historie der Datenbankzugriffe in einer Tabelle verfolgen. Für UDS/SQL-Anwendungen, die die COBOL-DML nutzen, ist auch diese Art der Darstellung möglich.

Daneben bietet die TRACE-Funktion für UDS/SQL mit der Anzeige aller Currency-Tabellen einen tiefen Einblick in die internen Zusammenhänge der UDS/SQL-Datenbank, die für die Programmierung und Fehleranalyse von entscheidender Bedeutung sind.

16 Meldungen des COBOL2000-Systems

Der COBOL2000-Compiler und das COBOL2000-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 in einer COBOL-Übersetzungseinheit 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 Übersetzungseinheitzeile, 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 COBOL2000-Compilers kann mit COMOPT PRINT-DIAGNOSTIC-MESSAGES=YES bzw. über die SDF-Option COMPILER-ACTION=PRINT-MESSAGE-LIST angefordert werden (siehe Abschnitte „Tabelle der COMOPT-Operanden" bzw. „COMPILER-ACTION-Option").

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 in der Übersetzungseinheit 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. COBOL2000 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 COBOLSprachelemente erkannt, auf die der Anwender zwar

		aufmerksam gemacht werden soll, die jedoch nicht die Ausgabe einer Warnungs- oder Fehlermeldung rechtfertigen.
0	Warnungsmeldung	Möglicherweise wurde in der Übersetzungseinheit 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 37: Fehlerklassen und ihre Bedeutung

2. Meldungen
   - die der Compiler über den Ablauf und die Beendigung des Übersetzungslaufes generiert (CBL90nn),
   - die das COBOL2000-Laufzeitsystem über den Ablauf und die Beendigung des Anwenderprogramms erzeugt (COB91nn).
   - des POSIX-Treibers für COBOL (CBL92nn)
   - die bei objektorientiertem Programmieren auftreten(COB93nn).

   Sie werden während der Übersetzung bzw. des Programmablaufs nach SYSOUT ausgegeben und haben folgenden Aufbau:

   ```
   { CBL90nn | COB91nn | CBL92nn | COB93nn } Text
   ```

   Dabei bezeichnet

   CBL90nn
   COB91nn
   CBL92nn
   COB93nn
   : die Kennnummer 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 Pro gramm 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 Übersetzungseinheitzeile enthält, bei deren Ausführung die Meldung ausgegeben wird.

Die (im Verlauf der Übersetzung ausgegebenen) Meldungen CBL9004, CBL9017, CBL9095, CBL9097 und CBL9099 werden unterdrückt, wenn vor dem Aufruf des Compilers der Auftragsschalter 4 gesetzt wird.

Mit dem Kommando HELP-MSG-INFORMATION können Sie zu einer Meldung den vollständigen Meldungstext anzeigen.

Er umfasst insbesondere folgende Information:

- Typ (der Meldung)
- Bedeutung (der Variablen in der Meldung)
- Verhalten (des Programms)
- ggf. Maßnahme (des Anwenders)

17 Anhang

In diesem Kapitel werden folgende Themen behandelt:

- Aufbau des COBOL2000-Systems
- Datenbankbedienung (UDS/SQL)
- Beschreibung der Listen

17.1 Aufbau des COBOL2000-Systems

Das COBOL2000-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 COBOL2000 sind im Common RunTime Environment (CRTE) enthalten (siehe CRTE-Benutzerhandbuch [2]).

17.1.1 Aufbau des COBOL2000-Compilers

Der COBOL2000-Compiler besteht aus einer Anzahl von Modulen, die linear gebunden sind.

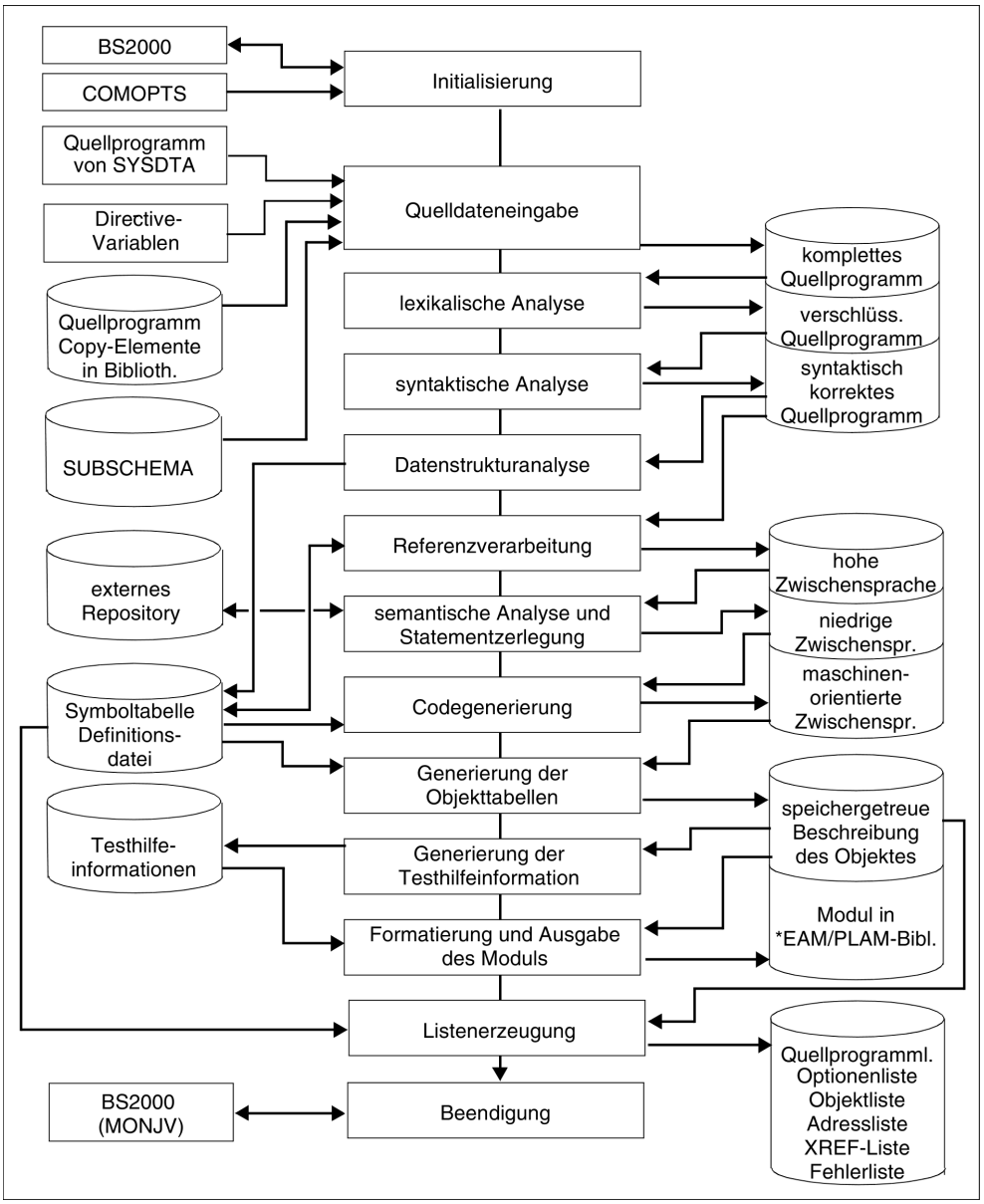
Die einzelnen Module bilden Funktionseinheiten, die durch den Ablauf einer COBOL-Übersetzung und durch die Einteilung eines COBOL-Programms in die einzelnen DIVISIONS vorgegeben werden.

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



17.1.2 Das COBOL2000-Laufzeitsystem

Das COBOL2000-Laufzeitsystem ist Bestandteil des Common RunTime Environment

(CRTE), der gemeinsamen Laufzeitumgebung für COBOL2000- und C/C++-Programme.

Das CRTE ist in einem eigenen CRTE-Benutzerhandbuch [2] beschrieben. Die COBOL2000-Laufzeitroutinen stellen dem COBOL2000-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 dem Handbuch „COBOL2000-Sprachbeschreibung“ [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/SQL
- Anschluss der COBOL-Programme an Ablaufteil-Funktionen

In folgender Tabelle sind die Namen und Funktionen der COBOL2000-Laufzeitmodule aufgeführt. In der Tabelle nicht enthalten sind diejenigen Laufzeitmodule, die nur aus Kompatibilitätsgründen noch im COBOL2000-Laufzeitsystem vorhanden sein müssen, sowie diejenigen Module, die für Zugriffe auf das POSIX-Dateisystem verwendet werden.

Name	Function
ITCMADPT *)	Adaptermodul bei Partial-Bind-Technik
ITCMAID1 *)	AID-Anschlussmodul (Datenteil)
ITCMECE1 *)	ENTRY, CANCEL, EXIT Arbeitsbereich
ITCMECE2 *)	Tabelle für COMOPT OPTIMIZE-CALL-IDENTIFIER bzw. SDF CALL-IDENTIFIER =OPTIMIZE
ITCMERF1 *)	Fehleranalyseroutine für Ein-/Ausgabe
ITCMINIT *)	ILCS-Initialisierung
ITCMMAT1 *)	Daten für mathematische (IML...-) Funktionen
ITCMMDP0 *)	OCCURS DEPENDING (rekursiv)
ITCMMEM1	ALLOCATE- und FREE-Arbeitsbereich
ITCMOBAS *)	OO: BASE KLASSE

ITCMOWK0 *)	OO: Arbeitsbereich für OO Laufzeitroutinen
ITCMPOVH *)	COBOL2000 Programm-Manager
ITCMSMG0 *)	SORT-/MERGE-Anweisung
ITCMSTBO *)	Tabellen sortieren
ITCMTOM0 *)	TOM Adapter Routine
ITCMXCAB *)	XML: Callback-Adapter für Basis-Funktionen
ITCMXCAS *)	XML: Callback-Adapter für XML PARSE-Anweisung
ITCMXDT1 *)	Datenmodul für Ausnahmebehandlung
ITCMXHC2 *)	XHCS Konvertierungen Arbeitsbereich
ITCMXMD1 *)	XML: Datenbereich für XML PARSE-Sonderregister
ITCMXWK0 *)	XML: Arbeitsbereich für XML-Laufzeitroutinen
ITCRACA0	ACCEPT-Anweisung
ITCRACX0	ACCEPT-Anweisung für Umgebungsvariable/Kommandozeile
ITCRAID2	AID-Anschlussmodul (Prozedurteil)
ITCRBCT0	Binäre Konstantentabelle
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 zeilensequenzielle Dateien
ITCRCLR0	CLOSE-Anweisung für relative Dateien
ITCRCLS0	CLOSE-Anweisung für sequenzielle Dateien
ITCRCLX0	CLOSE-Anweiung für XML-Dateien
ITCRCMD0	Ausführen eines BS2000-Kommandos
ITCRCNA0	Vergleich nationale figurative Konstante
ITCRCVB0	Umwandlung gepackt dezimal nach binär (10 bis 18 Ziffern)
ITCRCVD0	Umwandlung binär nach gepackt dezimal (10 bis 18 Ziffern)
ITCRCVF0	Umwandlung von/nach Gleitpunkt
ITCRCVL0	Umwandlung gepackt + dezimal von/nach binär (>18 Ziffern)

ITCRDFE0	Division extern Gleitpunkt
ITCRDPL0	Division von Dezimalzahlen > 15 Stellen
ITCRDSA0	DISPLAY-Anweisung
ITCRDSI1	Speicherzuweisung DYNAMIC-Daten
ITCRDSX0	DISPLAY-Anweisung für Umgebungsvariable
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)
ITCREPL2	Ausgeben Zeile mit Vorschub-Steuerzeichen
ITCREV0	Ereignisbehandlung (Rückkehr aus einem fremdsprachigen Unterprogramm)
ITCREV1	Ereignisbehandlung („recoverable interrupts“)
ITCREV2	Ereignisbehandlung („unrecoverable interrupts“)
ITCREV3	Ereignisbehandlung (übrige Ereignisse)
ITCRFAT0	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, DISPLAY-OF, NATIONAL-OF
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 zeilensequenzielle Dateien
ITCRIFR1	ISAM-FCB-Generierung für relative Dateien
ITCRIFS1	SAM-FCB-Generierung für sequenzielle Dateien
ITCRIFX1	FCB-Generierung für XML-Dateien

ITCRINI0	INITIALIZE-Anweisung
ITCRINS0	INSPECT-Anweisung
ITCRLHS2	Benutzerkennsatzbehandlung für sequenzielle Dateien
ITCRLNL1	LINAGE-Klausel bei WRITE für zeilensequenzielle Dateien
ITCRLNS1	LINAGE-Klausel bei WRITE für satzsequenzielle Dateien
ITCRMAT0	Verbindungsmodul zu den mathematischen (IML...-) Funktionen
ITCRMEM0	ALLOCATE- und FREE-Anweisung
ITCRMEV2	Unterbrechungsmeldung für Event-Handling-Routine
ITCRMPL0	Multiplikation von Dezimalzahlen > 15 Stellen
ITCRMSG0	Ausgabe von Fehlermeldungen, level 0
ITCRMSG3	Ausgabe von Fehlermeldungen
ITCRMVE0	MOVE für numerisch-druckaufbereitete Felder
ITCRNED0	Deeditierender MOVE
ITCRNSP0	CALL, CANCEL, ENTRY, EXIT im geschachtelten Programm
ITCROCA0	Prüfen Übereinstimmung aktuelle / formale Methodenparameter
ITCROFP2	OO: Formale Parameterbeschreibung bereitstellen
ITCROIF1	OO: Initialisierungsroutine für Dateien in Objekten
ITCROIS0	OO: Initialisierungsroutine für Klassen / Interfaces
ITCROMD0	OO: Prüfroutine für Objektsicht (object- view)
ITCROMS1	OO: Ausgabe von OO-Fehlermeldungen
ITCRONW1	OO: Neues Objekt erzeugen und initialisieren
ITCROOI0	Steuerroutine für Objektorientierte Initialisierung und Beendigung
ITCROPI0	OPEN-Anweisung für indizierte Dateien
ITCROPL0	OPEN-Anweisung für zeilensequenzielle Dateien
ITCROPR0	OPEN-Anweisung für relative Dateien
ITCROPS0	OPEN-Anweisung für sequenzielle Dateien
ITCROPX0	OPEN-Anweiung für XML-Dateien
ITCROSM0	OO: Methode auswählen
ITCROTC2	OO: Konformitätstest
ITCROVC1	OO: Prüfen von Interface Konformität
ITCROVI2	OO: Prüfen von Klassenvererbung
ITCRPAM1	physische Lese-/Schreibroutine für relative Dateien (PAM)
ITCRPBND	Partial-Bind-Großmodul

ITCRPCA0	Vergleiche unter PROGRAM COLLATING SEQUENCE
ITCRPCS0	Vergleiche unter PROGRAM COLLATING SEQUENCE
ITCRRCH0	Überprüfung von Tabellengrenzen
ITCRRDI0	READ-/START-Anweisung für indizierte Dateien
ITCRRDL0	READ-/START-Anweisung für zeilensequenzielle Dateien
ITCRRDR0	READ-/START-Anweisung für relative Dateien
ITCRRDS0	READ-Anweisung für sequenzielle Dateien
ITCRRDX0	READ-/START-Anweiung für XML-Dateien
ITCRRPW0	REPORT-WRITER-Steuermodul
ITCRSCH0	SEARCH-ALL-Anweisung
ITCRSEG0	Ansprung segmentierter COBOL-Programme
ITCRST11	CODE SET-Tabelle für ASCII
ITCRST21	CODE SET-Tabelle für ISO-7
ITCRSTG0	STRING-Anweisung
ITCRSTP0	STOP literal-Anweisung
ITCRTCA1	Klassentest-Tabelle für Test auf ALPHABETIC
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
ITCRTCN0	Klassentest für nationale Operanden
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
ITCRTOD3	Tageszeit / Datum (SVC-frei)
ITCRUDS0	DML-Link zum Database Handler
ITCRUPC0	Abwicklung Declaratives
ITCRUPC1	Abwicklung Declaratives
ITCRUPC2	Abwicklung Declaratives
ITCRUST0	UNSTRING-Anweisung
ITCRVCL0	Vergleich für Felder variabler Länge/Adresse oder > 256 Byte
ITCRVCN0	Vergleich nationaler Felder variabler Länge/Adresse oder >256 Byte

ITCRVMA0	MOVE ALL literal
ITCRVMN0	MOVE für nationale Felder variabler Länge/Adresse oder > 256 Byte
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 zeilensequenzielle Dateien
ITCRWRR0	WRITE-/REWRITE-Anweisung für relative Dateien
ITCRWRS0	WRITE-Anweisung für sequenzielle Dateien
ITCRXBND	XML: Parser-Nachladeroutine
ITCRXCFB	XML: Callback-Funktionen für Basis-Funktionalität
ITCRXCFS	XML: Callback-Funktionen für XML PARSE-Anweisung
ITCRXDP1	XML: Dokument-Baum-Verarbeitung
ITCRXFS0	Erweiterter File Status
ITCRXGE0	XML GENERATE-Anweisung
ITCRXHC1	XHCS Konvertierungen
ITCRXIT0	FILE STATUS und Fehlerbehandlungsroutine
ITCRXLC0	Ausnahmebehandlung
ITCRXPA0	XML PARSE-Anweisung
ITCRXPC1	XML: Dokument-Datei-Verarbeitung
ITCRXPF0	Potenzierung (Gleitpunkt)
ITCRXPI0	Potenzierung (Ganzzahl)

*) Modul nicht gemeinsam benutzbar

17.2 Datenbankbedienung (UDS/SQL)

In COBOL2000-BC nicht unterstützt !

Eine Beschreibung des universellen Datenbanksystems UDS/SQL findet sich in den Handbüchern „Entwerfen und Definieren“ [13], „Aufbauen und Umstrukturieren“ [14], „Anwendungen programmieren“ [15].

UDS/SQL-Datenbanken werden von Anwenderprogrammen bedient über

- COBOL-DML-Sprachelemente (DML ist integraler Bestandteil von COBOL) und
- 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/SQL-Anwenderprogramms kurz dargestellt.

Der Database Handler (DBH) als Kernkomponente des UDS/SQL-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 im Handbuch „Anwendungen programmieren“ [15] beschrieben.

schema-name/subschema-name werden bei der Schema- bzw. Subschema-Generierung festgelegt.

Übersetzen eines COBOL-DML-Programms

Der COBOL2000-Compiler erzeugt aus einem COBOL-DML-Programm ein ProgrammModul und ein Subschema-Modul. Bei der Kompilierung des Anwenderprogramms muss der COBOL-Compiler die COSSD-Datei der betroffenen Datenbank lesen. Dazu gibt es folgende Möglichkeiten:

1. Die COSSD-Datei wird dem COBOL-Compiler explizit zugewiesen mit dem Kommando

```
/ADD-FILE-LINK       LINK-NAME=UDSCOSSD, -
/                     FILE-NAME=[:catid:][$userid.]dbname.COSSD
```

Dabei sind `:catid:` und `$userid` die Katalogkennung und Benutzerkennung, unter der die COSSD-Datei katalogisiert ist. Ohne die Angabe `:catid:` bzw. `$userid` wird der Dateiname nach den Standardregeln des BS2000 komplettiert.

Die COSSD-Datei muss unter dem im Kommando angegeben Namen katalogisiert sein, da im Fehlerfall nicht nach einer COSSD-Datei an anderer Stelle gesucht wird. Dieses Verfahren ist zwingend erforderlich, wenn in allen Katalogen, die lokal von der Benutzerkennung aus zugreifbar sind, mehrere COSSD-Dateien mit dem entsprechenden Datenbanknamen existieren.

Beispiel für eine Kommandofolge:

```
/ADD-FILE-LINK UDSCOSSD,dbname.COSSD
/START-PROGRAM $COBOL2000
COMOPT MODULE=modulbibliothek
END Übersetzungseinheitdatei
```

2. Dem COBOL-Compiler wird der Datenbankname mitgeteilt mit dem Kommando

```
/SET-FILE-LINK       LINK-NAME=DATABASE, -
/                     FILE-NAME=[:catid:][$userid.]dbname
```

Die Angabe einer `:catid:` beim Kommando SET-FILE-LINK wird ignoriert. Der COBOL-Compiler sucht dann eine COSSD-Datei mit dem Namen `dbname.COSSD` in allen Katalogen, die lokal von derjenigen Benutzerkennung aus zugreifbar sind, die beim Kommando SET-FILE-LINK explizit angegeben wurde oder vom BS2000 ergänzt wurde. Dieses Verfahren kann nur verwendet werden, wenn im genannten Katalogumfeld nur eine COSSD-Datei mit dem entsprechenden Datenbanknamen existiert.

Beispiel für eine Kommandofolge:

```
/SET-FILE-LINK DATABASE,dbname
/START-PROGRAM $COBOL2000
COMOPT MODULE=modulbibliothek
END Übersetzungseinheitdatei
```

Falls ADD-FILE-LINK oder SET-FILE-LINK-Kommandos sowohl für LINK=DATABASE als auch für LINK=UDSCOSSD vorliegen, wird nur das Verfahren für LINK=UDSCOSSD angewandt.

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/SQL-Connection-Modul mit einzubinden ist (siehe hierzu [15]).

Beispiel eines Binderlaufs:

```
/START-BINDER
//START-LLM-CREATION INT-NAME=programmname
//INCLUDE-MODULES LIB=modulbibliothek, ELEM=cobol-dml-programm
//INCLUDE-MODULES LIB=udsmodulbibliothek, ELEM=uds-connection-modul
```

```
//RESOLVE-BY-AUTOLINK LIB=$.SYSLNK.CRTE
//SAVE-LLM LIB=modulbibliothek, ELEM=uds-test-prog
//END
```

Ablauf eines UDS/SQL-Anwenderprogramms

Der Ablauf eines UDS/SQL-Anwenderprogramms setzt bei Einsatz des independent DBH eine UDS/SQL-Session voraus. Die Verbindung zu dieser Session bzw. zur Datenbank stellt das SET-FILE-LINK-Kommando her (siehe Handbuch „UDS/SQL (BS2000/OSD)“ [15]).

Ablauf mit linked-in DBH:

```
/SET-FILE-LINK DATABASE,dbname
/START-PROGRAM dateiname
[DBH-Parameter]
PP END
[Anwenderprogramm-Parameter]
```

Ablauf mit independent DBH:

```
/SET-FILE-LINK DATABASE,dbname
/START-PROGRAM dateiname
[Anwenderprogramm-Parameter]
```

17.3 Beschreibung der Listen

In diesem Abschnitt werden anhand eines Programmbeispiels die Formate folgender Listen kurz erläutert, die COBOL2000 im Verlauf einer Übersetzung ausgibt:

- Steueranweisungsliste
- Übersetzungseinheitliste
- Adress-/Querverweisliste
- Fehlermeldungsliste

In den einzelnen Datensätzen einer Liste sind aus Gründen der Platzersparnis die Leerzeichen nach dem letzten gedruckten Zeichen entfernt.

17.3.1 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)
COBOL2000 V01.4A02	KOPIEREN	LIBRARY LISTING	09:58:34	2006-08-04	PAGE 0003

17.3.2 Steueranweisungsliste

Hier protokolliert COBOL2000

(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.

```
COBOL2000  V01.5A00  KOPIEREN                        COMOPT LISTING              09:58:34 2009-02-12  PAGE 0001


ENVIRONMENT           (1)
-----------
                          PROCESSOR                     : 7.500- S170-30
                          OPERATING SYSTEM              : BS2000 V16.0
                          COMPILER                      : COBOL2000  V01.5A00
                          TASK-SEQUENCE NUMBER          : 1k59
                          USER-ID                       : CAC21
                          Copyright (C) Fujitsu Technology Solutions 2009
                                        All Rights Reserved


OPTIONS IN EFFECT     (2)
-----------------
                          MODULE                        = COB
                          SYSLIST                       = (OPTIONS,DIAG,MAP,SOURCE,XREF)
                          GENERATE-LLM                  = YES
                          SOURCE-ELEMENT                = KOPIEREN.COB
                          MERGE-REFERENCES              = YES
                          MERGE-DIAGNOSTICS             = YES
                          ENABLE-XML-PROCESSING         = NO


OPTIONS BY DEFAULT    (3)
------------------
                          CHECK-DATE                    = YES
                          EXPAND-COPY                   = YES
                          LINE-LENGTH                   = 132
                          ALIGN-LLM-PAGE                = YES
                          LINES-PER-PAGE                = 064
                          MODULE-ELEMENT                = *STD
                          MODULE-VERSION                = *UPPER-LIMIT
                          SOURCE-VERSION                = *HIGHEST-EXISTING
                          EXPAND-SUBSCHEMA              = YES
                          MINIMAL-SEVERITY              = I
                          REPLACE-PSEUDOTEXT            = YES
                          RESET-PERFORM-EXITS           = YES
                          CONTINUE-AFTER-MESSAGE        = YES
                          GENERATE-INITIAL-STATE        = YES
                          DEFAULT-CALL-CONVENTION       = COMPATIBLE
                          INHIBIT-BAD-SIGN-PROPAGATION  = YES


FOR CUSTOMER SERVICE  (4)
--------------------
REV# = A
REV# = B
```

17.3.3 Übersetzungseinheitliste

Jede Zeile einer Übersetzungseinheitliste ist in die folgenden Bereiche unterteilt:

(1) Anzeigenfeld

Spalte 1 informiert über Fehler innerhalb der vom Benutzer vergebenen Nummerierung der Eingabesätze (Anzeige S) und über Verstöße gegen die maximale Zeilenlänge von 80 Zeichen beim Fixed-Format bzw. 248 Zeichen beim Free-Format (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.

Ein Minuszeichen (-) in Spalte 1 kennzeichnet Zeilen, die auf Grund von Compiler-Direktiven ignoriert wurden.

(2) Folgenummernfeld

Enthält eine von COBOL2000 vergebene, maximal 5-stellige Nummer, die zur Kennzeichnung des eingegebenen Übersetzungseinheit-Satzes dient. Diese Nummer dient zur eindeutigen Identifizierung der Quellcodezeilen. Sie findet sich in allen von COBOL2000 erzeugten Listen als Querverweisnummer wieder und wird zur Verknüpfung mit etwaigen Fehlermeldungen verwendet. Der maximale Wert beträgt 65535. Überschreitet eine Übersetzungseinheit diese Zahl, wird wieder von 0 an nummeriert.

(3) Zu Beginn jeder Seite einer Übersetzungseinheitliste 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 Programmzeilen

(5) Übersetzungseinheitbereich

Enthält den vom Benutzer eingegebenen Satz. Dabei ist zu beachten, dass nur abdruckbare Zeichen dargestellt werden.

Die folgenden Anteile sind nur in einer 'verdichteten' Liste vorhanden (siehe Parameter SOURCE=YES(CROSS-REFERENCE=YES) im Abschnitt „LISTING-Option".

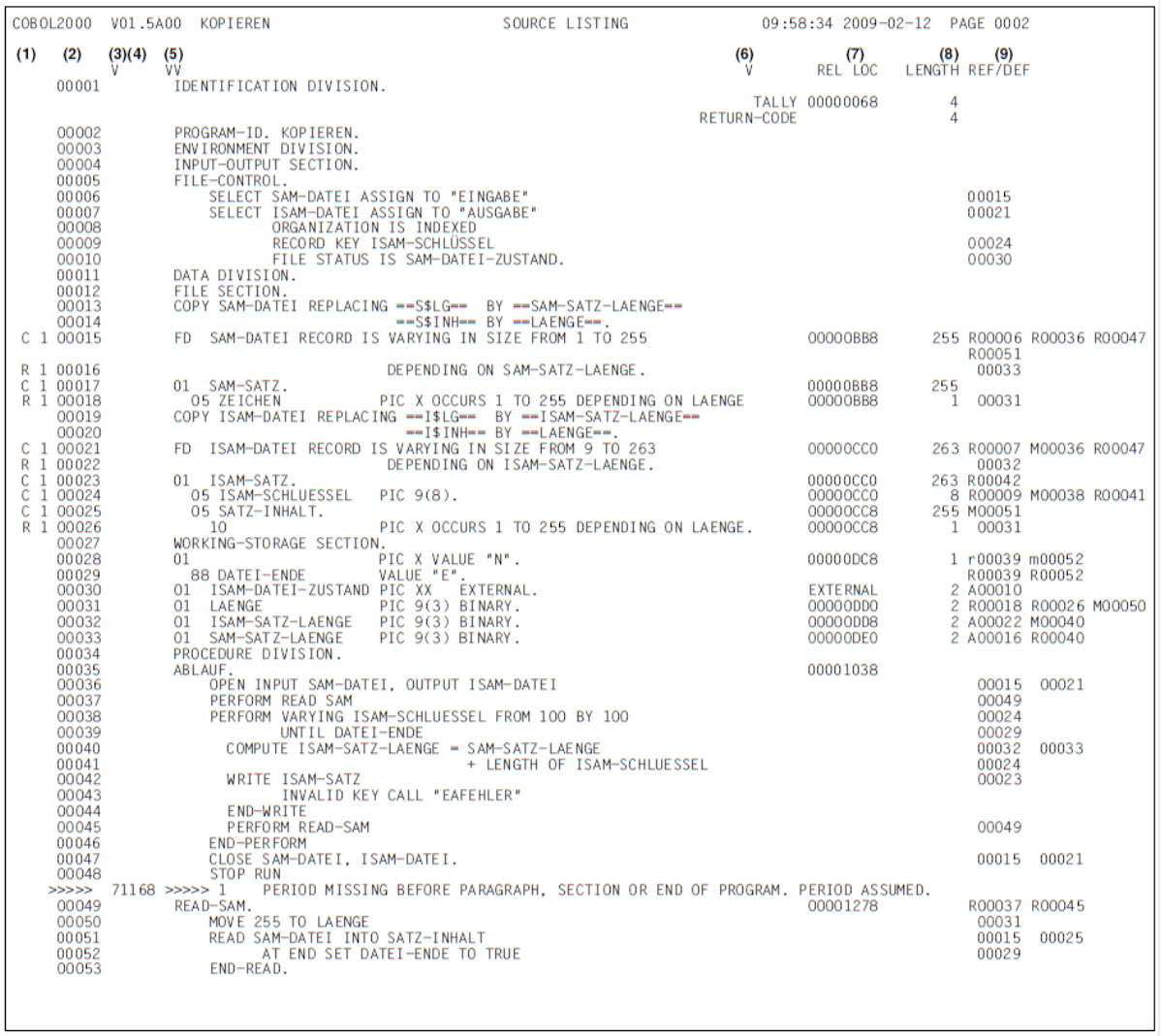
(6) Enthält eine Zeile mehr als eine Definition oder kommen in einer Übersetzungseinheit implizite Definitionen vor, dann werden diese im verdichteten Listing in zusätzlichen Zeilen dargestellt, in denen an Stelle des Quelltexts rechtsbündig nur der Name dieser Definition steht.

(7) REL LOC
enthält die Position einer Datendefinition bzw. eines Kapitel oder Paragrafennamens relativ zum Modulanfang.

(8) LENGTH
enthält die (dezimale) Länge des Bereichs im Modul, der einer Datendefinition zu geordnet wurde.

(9) REF/DEF
enthält die Folgenummern der Zeilen, die auf eine Definition Bezug nehmen, zu sammen mit der Art dieser Referenz (Erläuterung der Referenzart siehe Abschnitt „Adressliste") sowie umgekehrt beim Bezugnehmer die Folgenummer der Definitionszeile. Treten mehr Querverweise auf, als in eine Zeile passen, werden Fortsetzungszeilen gebildet (siehe auch Parameter LINE-SIZE im Abschnitt „LISTING-Option").

Im Listenbeispiel sind die Übersetzungsmeldungen „eingemischt“ (siehe Parameter INSERT-ERROR-MSG im Abschnitt „LISTING-Option").



```
COBOL2000  V01.5A00  KOPIEREN                        SOURCE LISTING              09:58:34 2009-02-12  PAGE 0002

(1)   (2)   (3)(4) (5)                                                                (6)          (7)        (8)   (9)
            V     VV                                                                   V        REL LOC    LENGTH REF/DEF
      00001       IDENTIFICATION DIVISION.
                                                                                   TALLY 00000068        4
                                                                                RETURN-CODE              4
      00002       PROGRAM-ID. KOPIEREN.
      00003       ENVIRONMENT DIVISION.
      00004       INPUT-OUTPUT SECTION.
      00005       FILE-CONTROL.
      00006           SELECT SAM-DATEI ASSIGN TO "EINGABE"                                                  00015
      00007           SELECT ISAM-DATEI ASSIGN TO "AUSGABE"                                                 00021
      00008                  ORGANIZATION IS INDEXED
      00009                  RECORD KEY ISAM-SCHLÜSSEL                                                     00024
      00010                  FILE STATUS IS SAM-DATEI-ZUSTAND.                                             00030
      00011       DATA DIVISION.
      00012       FILE SECTION.
      00013       COPY SAM-DATEI REPLACING ==S$LG==  BY ==SAM-SATZ-LAENGE==
      00014                                ==S$INH== BY ==LAENGE==.
  C 1 00015       FD  SAM-DATEI RECORD IS VARYING IN SIZE FROM 1 TO 255                 00000BB8      255 R00006 R00036 R00047
                                                                                                          R00051
  R 1 00016                     DEPENDING ON SAM-SATZ-LAENGE.                                             00033
  C 1 00017       01  SAM-SATZ.                                                        00000BB8      255
  R 1 00018         05 ZEICHEN          PIC X OCCURS 1 TO 255 DEPENDING ON LAENGE     00000BB8        1  00031
      00019       COPY ISAM-DATEI REPLACING ==I$LG==  BY ==ISAM-SATZ-LAENGE==
      00020                             ==I$INH== BY ==LAENGE==.
  C 1 00021       FD  ISAM-DATEI RECORD IS VARYING IN SIZE FROM 9 TO 263                00000CC0      263 R00007 M00036 R00047
  R 1 00022                     DEPENDING ON ISAM-SATZ-LAENGE.                                            00032
  C 1 00023       01  ISAM-SATZ.                                                       00000CC0      263 R00042
  C 1 00024         05 ISAM-SCHLUESSEL   PIC 9(8).                                    00000CC0        8 R00009 M00038 R00041
  C 1 00025         05 SATZ-INHALT.                                                   00000CC8      255 M00051
  R 1 00026           10                PIC X OCCURS 1 TO 255 DEPENDING ON LAENGE.    00000CC8        1  00031
      00027       WORKING-STORAGE SECTION.
      00028       01                    PIC X VALUE "N".                            00000DC8        1 r00039 m00052
      00029         88 DATEI-ENDE       VALUE "E".                                                      R00039 R00052
      00030       01  ISAM-DATEI-ZUSTAND PIC XX   EXTERNAL.                          EXTERNAL        2 A00010
      00031       01  LAENGE           PIC 9(3) BINARY.                            00000DD0        2 R00018 R00026 M00050
      00032       01  ISAM-SATZ-LAENGE   PIC 9(3) BINARY.                          00000DD8        2 A00022 M00040
      00033       01  SAM-SATZ-LAENGE    PIC 9(3) BINARY.                          00000DE0        2 A00016 R00040
      00034       PROCEDURE DIVISION.
      00035       ABLAUF.                                                           00001038
      00036           OPEN INPUT SAM-DATEI, OUTPUT ISAM-DATEI                                       00015  00021
      00037           PERFORM READ SAM                                                              00049
      00038           PERFORM VARYING ISAM-SCHLUESSEL FROM 100 BY 100                               00024
      00039                   UNTIL DATEI-ENDE                                                      00029
      00040             COMPUTE ISAM-SATZ-LAENGE = SAM-SATZ-LAENGE                                  00032  00033
      00041                                      + LENGTH OF ISAM-SCHLUESSEL                        00024
      00042             WRITE ISAM-SATZ                                                             00023
      00043                   INVALID KEY CALL "EAFEHLER"
      00044             END-WRITE
      00045             PERFORM READ-SAM                                                            00049
      00046           END-PERFORM
      00047           CLOSE SAM-DATEI, ISAM-DATEI.                                                  00015  00021
      00048           STOP RUN
     >>>>>  71168 >>>>> 1    PERIOD MISSING BEFORE PARAGRAPH, SECTION OR END OF PROGRAM. PERIOD ASSUMED.
      00049       READ-SAM.                                                         00001278          R00037 R00045
      00050           MOVE 255 TO LAENGE                                                            00031
      00051           READ SAM-DATEI INTO SATZ-INHALT                                               00015  00025
      00052                AT END SET DATEI-ENDE TO TRUE                                            00029
      00053           END-READ.
```

Als zweiter Teil der Übersetzungseinheitliste 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:

(10) Folgenummer der Programmzeile, in der die COPY-Anweisung auftritt

(11) Linkname aus der COPY-Anweisung

(12) Bibliothekstyp

(13) Elementname

(14) Datum

(15) Versionsnummer, mit der das Bibliothekselement in der Bibliothek eingetragen ist. Datum und Versionsnummer sind nicht immer vorhanden.

(16) Dateiname, unter dem die Bibliothek im Dateisystem eingetragen ist.

```
COBOL2000  V01.5A00  KOPIEREN                      LIBRARY LISTING              09:58:34 2009-02-12 PAGE 0003

  (10)     (11)      (12)       (13)                    (14)          (15)                  (16)

 SOURCE LIBRARY- (LIB-) ELEMENT-NAME                   USER        VERSION                FILE-NAME
 SEQ-NO NAME     ORG                                   DATE

 SOURCE         PLAM   KOPIEREN.COB                2004-07-29 ~                  :2OSC:$CAC21.Manualbeispiel
 00013  COBLIB   PLAM   SAM-DATEI                   2004-07-29 ~                  :2OSC:$CAC21.Manualbeispiel
 00019  COBLIB   PLAM   ISAM-DATEI                  2004-07-29 ~                  :2OSC:$CAC21.Manualbeispiel
```

Besonderheiten der Übersetzungseinheitliste für Free-Format

Wesentliche, für den Benutzer sichtbare Unterschiede zwischen Fixed- und Free-Format betreffen das Listing.

Im Listing erfolgt ein Umbruch der Free-Format-Zeile zu Teilzeilen von bis je 80 Zeichen Länge. Diese Teilzeilenlänge ergibt sich aus der alten Zeilenlänge (Sequence-Area + Indikator-Area + Programmtext-Area + Comment-Area). Bei Free-Format handelt es sich über die gesamte Teilzeilenlänge um Programmtext.

```
...
...
     00079 *>       !         !         !         !         !         !         !         !
     00080
     00081          add 1 to a. add 1 to b. add 1 to c. add 1 to d. add 1 to e. add 1 to f.
     +      add 1 to g. add 1 to h. add 1 to i. add 1 to j. add 1 to k. add 1 to l. add 1 t
     +     o m. add 1 to n. add 1 to o. add 1 to p. add 1 to q. add 1 to r. add 1 to s. add
     +      1 to t.
     00082
     00083 *>       !         !         !         !         !         !         !         !
     00084
     00085
     +
     +                      add 1 to u. add 1 to v. add 1 to w. add 1 to x. add 1 to y. add
     +      1 to z.
     00086
     00087 *>       !         !         !         !         !         !         !         !
     00088
     ...
...
```

Zeilen, die nicht länger als 80 Zeichen sind, werden wie bisher unter voller Ausnutzung des verfügbaren Platzes aufgelistet. Eine Zeile mit einer Länge von 81 bis 160 Zeichen wird in eine Teilzeile mit 80 Zeichen und in eine kürzere Teilzeile mit 1 bis 80 Zeichen zerlegt. Letztere Teilzeile erhält zur Kennzeichnung keine vorangestellte Zeilennummer, sondern nur ein einzelnes „+“-Zeichen.
Zeilen mit einer Länge von 161 bis 240 Zeichen werden analog in drei Zeilen zerlegt. Zeilen mit einer Länge von 241 oder mehr Zeichen werden in vier Teilzeilen zerlegt. Dabei ist die vierte Teilzeile höchstens 8 Zeichen lang, da die maximale Zeilenlänge mit 248 Zeichen begrenzt ist.

In dieser Form des Listings stehen die Teilzeilen direkt untereinander.

Wird das verdichtete Listing (siehe Abschnitt „LISTING-Option") eingeschaltet, sieht derselbe Listingausschnitt folgendermaßen aus:

```
...
...
     00079 *>       !         !         !         !         !         !         !         !
     00080
     00081          add 1 to a. add 1 to b. add 1 to c. add 1 to d. add 1 to e. add 1 to f.    00014  00015  00016
                                                                                              00017  00018  00019
     +      add 1 to g. add 1 to h. add 1 to i. add 1 to j. add 1 to k. add 1 to l. add 1 t    00020  00021  00022
                                                                                              00023  00024  00025
     +     o m. add 1 to n. add 1 to o. add 1 to p. add 1 to q. add 1 to r. add 1 to s. add    00026  00027  00028
                                                                                              00029  00030  00031
                                                                                              00032
     +      1 to t.                                                                          00033
     00082
     00083 *>       !         !         !         !         !         !         !         !
     00084
     00085
     +
     +                      add 1 to u. add 1 to v. add 1 to w. add 1 to x. add 1 to y. add    00034  00035  00036
                                                                                              00037  00038
     +      1 to z.                                                                          00039
     00086
     00087 *>       !         !         !         !         !         !         !         !
     00088
     ...
...
```

Da die XREF-Ausgaben gelegentlich den Einschub zusätzlicher Leerzeilen erzwingen, ist nur durch das „+ “Zeichen am Zeilenanfang erkennbar, ob eine solche eingeschobene Zeile wegen der XREF-Einträge generiert wurde, oder ob sie tatsächlich eine echte Teilzeile ist, die zufällig nur Leerzeichen enthält (siehe Beispiel Zeile 00085).

Die Zuordnung von XREF-Einträgen zu den entsprechenden Teilzeilen bezieht sich immer auf das erste Zeichen eines Datennamens.

Der Umbruch einer Free-Format-COBOL-Zeile führt nicht immer zu einem übersichtlichen Listing. Daher ist es sinnvoll, durch geeignete Gruppierung der Programmtext-Elemente ein übersichtlicheres Listing-Bild zu erreichen, indem z.B. die Programmtext-Elemente auf gedachten Tabulator-Positionen (20, 40 oder 80 Zeichen) ausgerichtet werden. In vielen Fällen ist es angebracht, sich auf eine Zeilenlänge von 80 Zeichen zu beschränken.

17.3.4 Die Formatsteueranweisungen TITLE, EJECT, SKIP

Der COBOL2000-Compiler unterstützt die Formatsteueranweisungen TITLE, EJECT und SKIP. Mit diesen Anweisungen in der Übersetzungseinheit kann das Aussehen der Übersetzungseinheitliste beeinflusst werden.

Für alle Formatsteueranweisungen gilt:

- Sie dürfen nicht mit einem Punkt abgeschlossen werden.
- Sie müssen allein ab Spalte 12 in einer Zeile stehen.
- Sie erscheinen selbst nicht in der Übersetzungseinheitliste.

TITLE-Anweisung

Funktion

Die Anweisung bewirkt, dass nachfolgend in den Kopfzeilen der Übersetzungseinheitliste 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

```
TITLE literal
```

Regel

literal muss ein maximal 53 Zeichen langes nichtnumerisches Literal sein.

EJECT-Anweisung

Funktion

Die Anweisung bewirkt, dass der nachfolgende Text der Übersetzungseinheitliste 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 Übersetzungseinheitliste 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 17-1

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 Übersetzungseinheitliste steht „WORKING-STORAGE SECTION“

(2) In der Kopfzeile der nächsten Seite(n) der Übersetzungseinheitliste steht „PROCEDURE DIVISION“.

(3) Der nachfolgende Text (ANFANG SECTION...) beginnt auf der nächsten Seite.

(4) Vor dem nachfolgenden Text (ENDE SECTION) stehen drei Leerzeilen.

17.3.5 Fehlermeldungsliste

Die von COBOL2000 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 Übersetzungseinheitzeile an, in der der Fehler auftrat.

(2) MSG INDEX — gibt die Fehlermeldungskennzeichnung an.

(3) SEVERITY CODE — gibt die Fehlerklasse an (siehe Tabelle 37 in Kapitel "Meldungen des COBOL2000-Systems").

(4) ERROR MESSAGE — enthält den erklärenden Text und gegebenenfalls die von COBOL2000 durchgeführte Korrektur oder einen von COBOL2000 angenommenen Standardwert.

Am Ende der Fehlermeldungsliste wird eine Abschlussinformation über Gesamtanzahl aller aufgetretenen Fehler sowie Gesamtanzahl der Fehler in den verschiedenen Fehlerklassen ausgedruckt.

```
COBOL2000  V01.5A00  KOPIEREN                      DIAGNOSTIC LISTING          09:58:34 2009-02-12 PAGE 0007
  (1)       (2)       (3)          (4)

 SOURCE  MSG    SEVERITY
 SEQ-NO  INDEX      CODE    ERROR MESSAGE

  00048  71168        1     PERIOD MISSING BEFORE PARAGRAPH, SECTION OR END OF PROGRAM. PERIOD ASSUMED.
 TOTAL 00001 STATEMENTS IN THIS DIAGNOSTIC LISTING.
       00001 IN SEVERITY CODE 1
```

17.3.6 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 Übersetzungseinheitzeile, 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 Übersetzungseinheit-Zeilennummern in aufsteigender Reihenfolge, in denen Anweisungen stehen, die auf die Datendefinition Bezug nehmen.
Treten mehr Querverweise auf, als in die Zeile passen, werden Fortsetzungszeilen gebildet (siehe Parameter LINE-SIZE im Abschnitt „LISTING-Option").

Der Zeilennummer vorangestellt ist die Art der Bezugnahme:

M modify/schreibend

R read/lesend

A addressing/adressierend

Die entsprechenden Kleinbuchstaben zeigen implizite Zugriffe an (das betrifft zum Beispiel korrespondierende untergeordnete Felder bei MOVE CORRESPONDING oder das Datenfeld, auf das sich ein Bedingungsname bezieht).

(12) LVL
Schachtelungstiefe des Programms, beginnend bei 000 für das äußerste Programm.

(13) ÜBERSETZUNGSEINHEITNAME / SECTION NAME / PARAGRAPH NAME
Angabe des Übersetzungseinheitnamens und der darin vorhandenen Kapitel- und Paragrafennamen.

```
COBOL2000  V01.5A00  KOPIEREN                    LOCATOR MAP LISTING          09:58:34 2009-02-12  PAGE 0004

                                        DATA DIVISION           KOPIEREN     (1)
                                        FILE SECTION

                                  FILE NAME                 SAM-DATEI          (2)
                                  FILE SERIAL NO.           001
                                  ADDR LHE FCB              000003A8

  (3)      (4)       (5)       (6)       (7)  (8)                    (9)                  (10)         (11)

        MODULE    GROUP   POSITION
 SOURCE REL       REL     IN GROUP LEV                          LENGTH IN BYTES              REFERENCED
 SEQ-NO ADDR      ADDR    DEC       NO                          DEC        HEX      FORMAT   BY STATEMENTS

  00015                             FD SAM-DATEI                                              R00006 R00036 R00047 R00051
  00017 00000BB8 000000 00000001  01 SAM-SATZ                    0000000255 000000FF
  00018 00000BB8 000000 00000001  05 ZEICHEN                    0000000001 00000001 DISPLAY

                                  FILE NAME                 ISAM-DATEI
                                  FILE SERIAL NO.           002
                                  ADDR LHE FCB              00000790
        MODULE    GROUP   POSITION
 SOURCE REL       REL     IN GROUP LEV                          LENGTH IN BYTES              REFERENCED
 SEQ-NO ADDR      ADDR     DEC      NO                          DEC        HEX      FORMAT   BY STATEMENTS

  00021                             FD ISAM-DATEI                                             R00007 M00036 R00047
  00023 00000CC0 000000 00000001  01 ISAM-SATZ                   0000000263 00000107                R00042
  00024 00000CC0 000000 00000001  05 ISAM-SCHLUESSEL            0000000008 00000008 ZONED DEC R00009 M00038 R00041
  00025 00000CC8 000008 00000009  05 SATZ-INHALT                0000000255 000000FF           M00051
  00026 00000CC8 000008 00000009  10 FILLER                     0000000001 00000001 DISPLAY
```

```
COBOL2000  V01.5A00  KOPIEREN                    LOCATOR MAP LISTING          09:58:34 2009-02-12  PAGE 0005

                                        DATA DIVISION           KOPIEREN
                                        WORKING-STORAGE SECTION
        MODULE    GROUP   POSITION
 SOURCE REL       REL     IN GROUP LEV                          LENGTH IN BYTES              REFERENCED
 SEQ-NO ADDR      ADDR    DEC       NO                          DEC        HEX      FORMAT   BY STATEMENTS

  00001 00000068                  G77 TALLY                     0000000004 00000004 COMP
  00001                           G77 RETURN-CODE               0000000004 00000004 COMP-5
  00028 00000DC8 000000 00000001   01 FILLER                    0000000001 00000001 DISPLAY   r00039 m00052
  00029                            88 DATEI-ENDE                                              R00039 R00052
  00030 EXTERNAL 000000 00000001   01 ISAM-DATEI-ZUSTAND        0000000002 00000002 DISPLAY   A00010
  00031 00000DD0 000000 00000001   01 LAENGE                    0000000002 00000002 BINARY    R00018 R00026 M00050
  00032 00000DD8 000000 00000001   01 ISAM-SATZ-LAENGE           0000000002 00000002 BINARY    A00022 M00040
  00033 00000DE0 000000 00000001   01 SAM-SATZ-LAENGE            0000000002 00000002 BINARY    A00016 R00040
```

```
COBOL2000  V01.5A00  KOPIEREN                    LOCATOR MAP LISTING          09:58:34 2009-02-12  PAGE 0006

                                        PROCEDURE DIVISION

                  (12)      (13)                    (11)

                  LVL SOURCE UNIT NAME
 SOURCE REL             SECTION NAME                 REFERENCED
 SEQ-NO ADDR              PARAGRAPH NAME             BY STATEMENTS

                  000 KOPIEREN
  00035 00001040          ABLAUF
  00049 00001280          READ-SAM                   R00037 R00045
```

18 Literatur

Die Handbücher sind online unter http://manuals.ts.fujitsu.com zu finden oder in gedruckter Form gegen gesondertes Entgelt unter http://manualshop.ts.fujitsu.com zu bestellen.

[1] **COBOL2000** (BS2000/OSD))
COBOL-Compiler
Sprachbeschreibung

[2] **CRTE (BS2000/OSD)**
Common Runtime
EnvironmentBenutzerhandbuch

[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/OSD)
Benutzerhandbuch

[7] **JV** (BS2000/OSD)
Jobvariablen
Benutzerhandbuch

[8] **AID** (BS2000)
Advanced Interactive Debugger
Testen von COBOL-Programmen
Benutzerhandbuch

[9] BS2000
TSOSLNK
Benutzerhandbuch

[10] **BLSSERV**
Bindelader-Starter in BS2000/OSD
Benutzerhandbuch

[11] **LMS** (BS2000/OSD)
SDF-Format
Benutzerhandbuch

[12] **BS2000/OSD-BC**
Systeminstallation
Benutzerhandbuch

[13] **UDS/SQL** (BS2000/OSD)
Entwerfen und Definieren
Benutzerhandbuch

[14]

UDS/SQL (BS2000/OSD)
Aufbauen und Umstrukturieren
Benutzerhandbuch

[15] **UDS/SQL** (BS2000/OSD)
Anwendungen programmieren
Benutzerhandbuch

[16] **ESQL-COBOL** (BS2000/OSD)
ESQL-COBOL für SESAM/SQL-Server
Benutzerhandbuch

[17] **SESAM/SQL-Server** (BS2000/OSD)
SQL-Sprachbeschreibung Teil 1: SQL-Anweisungen
Benutzerhandbuch

[18] **SQL für UDS/SQL**
Sprachbeschreibung

[19] **EDT** (BS2000/OSD)
Anweisungen
Benutzerhandbuch

[20] **AID** (BS2000)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

[21] **AID** (BS2000/OSD)
Testen auf Maschinencode-Ebene
Benutzerhandbuch

[22] **BINDER**
Binder in BS2000/OSD
Benutzerhandbuch

[23] **openUTM** (TRANSDATA, BS2000)
Planen und entwerfen
Benutzerhandbuch

[24] **openUTM** (BS2000/OSD, UNIX, Windows)
Anwendungen programmieren mit KDCS für COBOL, C und C++
Benutzerhandbuch

[25] **openUTM** (BS2000/OSD)
Anwendungen generieren und betreiben
Benutzerhandbuch

[26] **openUTM** (BS2000/OSD, UNIX, Windows)
Anwendungen administrieren
Benutzerhandbuch

[27] **XML für openUTM**
Datamarshalling mit XML

[28] **SDF-P** (BS2000/OSD)
Programmieren in der Kommandosprache
Benutzerhandbuch

[29] **POSIX** (BS2000/OSD)
Kommandos
Benutzerhandbuch

[30] **POSIX** (BS2000/OSD)
Grundlagen für Anwender und Systemverwalter
Benutzerhandbuch

[31] **C/C++** (BS2000/OSD)
POSIX-Kommandos des C/C++-Compilers
Benutzerhandbuch

[32] **BS2000/OSD**
Softbooks Deutsch
CD-ROM

[33] **XHCS** (BS2000/OSD)
8-bit-Code- und Unicode-Unterstüzung im BS2000/OSD
Benutzerhandbuch

[34] **IMON** (BS2000/OSD)
Installationsmonitor
Benutzerhandbuch