JCL

Job Control Language

im

z/OS-Umfeld

praktisch anwenden

von:

Markus Karl Scheibe

Informatik-/Betriebswirt (VWA)
Nachtigalstr. 10
30173 Hannover
Tel.: 0173/6181348

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Verarbeitungsmethoden

Mit Hilfe der Informationsverarbeitung erhält man die Möglichkeit, die unterschiedlichsten Informationen zu verarbeiten. Man unterscheidet zwischen zwei Vorgehensweisen:

- Dialog-Betrieb, sogenannte Foreground-Verarbeitung
- Batch-Betrieb, sogenannte Background-Verarbeitung

Der Dialog-Betrieb orientiert sich am Anwender. Der Anwender kommuniziert direkt mit der IV-Anlage. Auf eine Eingabe folgen direkt die Verarbeitung und sofort danach die Ausgabe des Ergebnisses.

Im Gegensatz zum Dialog-Betrieb wird in der Batch-Verarbeitung (=Stapel-Verarbeitung) eine große Zahl von Eingaben in direkter Folge verarbeitet. Dabei besteht zwischenzeitlich keine Möglichkeit als Anwender in die Verarbeitung einzugreifen.

Im Batch-Betrieb sammelt der Anwender eine Reihe von Eingaben, welche er in einer sequentiellen Datei ablegt und auf einem Volume, dies können sowohl Platten als auch Magnetbänder oder Kassetten sein, abspeichert.

Danach wird jede der Eingaben einzeln verarbeitet. Dazu werden seine Daten vom Speichermedium geladen und ausgeführt bzw. interpretiert. Wurde dieser Abschnitt abgearbeitet, werden dessen Daten wieder gespeichert.

Dieser Verarbeitungsschritt wird solange durchgeführt, bis alle Datensequenzen in der Eingabedatei abgearbeitet sind. Wenn alle Arbeitsschritte beendet wurden, kann das Endergebnis auf einem Drucker oder Bildschirm ausgegeben werden.

Die Batchverarbeitung erfordert eine Sprache, die eine Kommunikation mit dem IV-System ermöglicht. Mit dieser Sprache kann der Anwender seine Wünsche und Vorstellungen dem IV-System näher bringen.

In der Welt des MVS-Betriebssystems ist das die Job Control Language, kurz JCL genannt. Sie ist der Schlüssel zum System.

1.2 Was ist ein (MVS-)Betriebssystem?

Das Betriebssystem ist das physikalische Basissystem einer EDV-Anlage in Softwareform. Es koordiniert das Zusammenspiel der einzelnen Applikationen und weist ihnen die benötigte CPU-Zeit zu. In der Standard-PC-Umgebung heißt das Betriebssystem Windows, in der IBM-Mainframe-Umgebung z/OS. Das Prinzip ist das Gleiche, nur dass der Mainframe über mehr CPU verfügt.

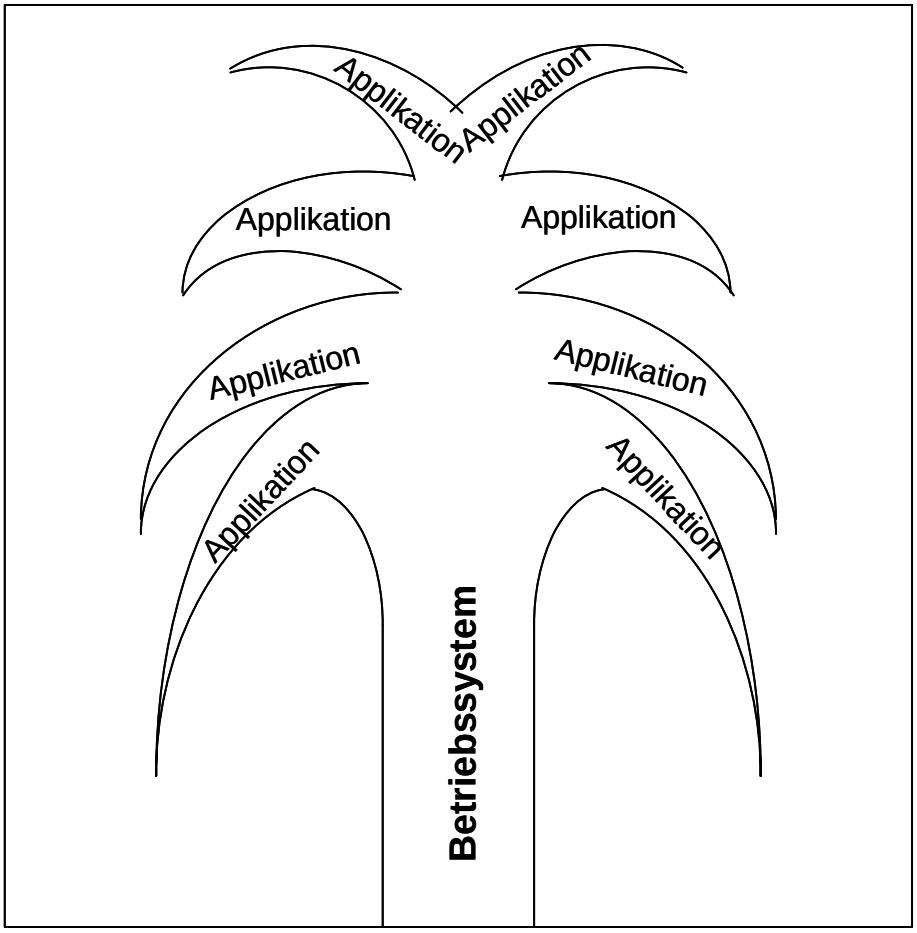

Abb. 1: Das Prinzip eines Betriebssystems

MVS ist die Abkürzung für „Multiple Virtual Storage“ und steht für mehrfacher, virtueller Speicher. MVS ist ein von der IBM in den 60er Jahren entwickeltes, UNIX-verwandtes Betriebssystem. Eingesetzt in Mainframes benötigt es geringe Hardwareanforderungen und überzeugt durch hohe Stabilität im Langzeitbetrieb.

1.3 Der Katalogeintrag

Ein Betriebssystem ist von der Basis aus gesehen mit einem konventionellen Buch vergleichbar. In einem Buch dient das Inhaltsverzeichnis dazu, sich einen Überblick darüber zu verschaffen, auf welchen Seiten gesuchte Textstellen zu finden sind. Das Inhaltsverzeichnis führt den Leser mit Hilfe von Kapitelüberschriften zur entsprechenden Buchseite.

Das Betriebssystem verfügt über ein Dateisystem. In der Welt von MS-DOS lautet der Name des Dateisystems 'FAT[1]', unter OS/390 'VTOC[2]'. Da Dateien in der Regel auf der Festplatte nicht zusammenhängend, sondern auf mehrere Spuren verteilt abgelegt werden, werden Informationen benötigt, mit deren Hilfe die einzelnen Dateisegmente wieder gefunden werden können. Diese Aufgabe übernimmt das Inhaltsverzeichnis.

[1] File allocation table
[2] Volume table of contents

Wenn eine Datei angelegt wird, im umgangssprachlichen spricht man hier von Datei-Allokation, wird für den Dateinamen durch die Disposition[3] 'CATLG' der Katalogeintrag veranlasst. Von nun an können alle Informationen, die in diese Datei geschrieben werden, über den Katalogeintrag zusammenhängend wieder gefunden und ausgegeben werden.

1.4 Was ist JCL?

JCL ist eine Sprache. Im EDV-Logo wird oft auch der Auftrag als JCL bezeichnet, obwohl hier der Ausdruck Job angebrachter wäre. Man wird jedoch kaum hören, dass jemand einen Auftrag schreibt, man schreibt JCL.

Häufig wird auch der Begriff „Job Control“ benutzt. Dann ist meistens eine Anzahl zusammenhängender Statements, also Anweisungen, gemeint. In der Regel ist dies ein kompletter Job oder Step.

Wenn man im täglichen Leben Aufträge in schriftlicher oder mündlicher Form erteilt, kann es leicht zu Missverständnissen kommen, da es im umgangssprachlichen unterschiedliche Interpretationsmöglichkeiten gibt. Diese Probleme gibt es bei der Job Control Language nicht. Anweisungen sind hier immer eindeutig. Entweder

⇨ eindeutig richtig oder
⇨ eindeutig falsch,

da JCL eine Sprache ohne sprachliche Freiheiten ist.

1.5 JCL im Vergleich zu anderen Programmiersprachen

Vergleicht man die JCL mit anderen Programmiersprachen wird man feststellen, dass JCL eine Sprache mit Eigenheiten ist, wie andere Programmiersprachen sie auch haben. Als Beispiel soll eine Variablen-Deklaration dienen:

JCL:
```
SET NAME=’SCHEIBE’
SET VORNAME=’MARKUS’
```

VBA:
```
DIM NAME AS STRING
NAME = „SCHEIBE“
DIM VORNAME AS STRING
VORNAME = „MARKUS“
```

Hier werden folgende Unterschiede sichtbar:

- Während im VBA-Code zuerst eine Meta-Information herausgegeben werden muss und erst im nächsten Schritt der Wert einer Variablen zugewiesen

[3] vgl. Kapitel 3.3

werden kann wird in der JCL der Wert direkt zugewiesen. Es werden keine Meta-Informationen benötigt.

- Im VBA-Code wird bei einer derartigen Wertzuweisung nicht mit SET gearbeitet.

2 Job-Aufbau

Um sich mit JCL-Statements vertraut zu machen ist es hilfreich, ihre Arbeitsweise kennenzulernen. Deshalb wird nachfolgend der Job-Aufbau kurz erläutert.

2.1 Job-Spezifikation

Wie bei jedem richtigen Auftrag muss auch hier zunächst einmal eine allgemeine Spezifikation des Auftrages oder Jobs erfolgen. Die Job-Spezifikation erfolgt durch das sogenannte „Job-Statement“.

Ein Job-Statement kann folgendermaßen aufgebaut sein:

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
```

Abb. 2: Job-Statement

Ein Job ist eine unabhängige Arbeitseinheit. Sie stellt sich als eine Gruppe von Arbeitsanweisungen dar, welche entweder von einem Bediener (Operator) eingegeben werden oder auf einem Speichermedium abgelegt sind und an den Rechner übertragen werden.

Die JCL-Statements steuern das Betriebssystem bei der Verarbeitung und beschreiben alle benötigten Ein- und Ausgabeeinheiten. Da es sich hier um komplexe Anweisungen handelt, werden diese als Dataset zusammengefasst und auf Speichermedien abgelegt. Dadurch wird der Benutzer in die Lage versetzt, den Namen des katalogisierten Dataset mit den Steueranweisungen aufzurufen ohne die einzelnen Steueranweisungen eingeben zu müssen

Die Bedeutung der einzelnen Angaben wird in den späteren Kapiteln erläutert, in denen auf die JCL explizit eingegangen wird.

2.2 Arbeitsabläufe

Innerhalb eines Jobs werden Arbeitsabläufe nacheinander beschrieben. Ein Job muss mindestens einen Arbeitsablauf beinhalten, es können aber auch mehrere sein. Hat ein Job keinen Arbeitsablauf, bricht er ab. Ein Arbeitsablauf wird durch ein sogenanntes „EXEC-Statement“ gekennzeichnet.

Ein Computer ist eine Einheit, die nach dem EVA-Prinzip arbeitet. Es werden Daten eingegeben, verarbeitet und das Ergebnis ausgegeben. Das EXEC-Statement dient dazu, diese Verarbeitungsprozesse zu beschreiben.

Ein EXEC-Statement kann folgendermaßen aussehen:

```
//STEP0001 EXEC PGM=IEBGENER
…
//STEP0002 EXEC PROC=PRDPROC1
//STEP0003 EXEC PRDPROC2
```

Abb. 3: EXEC-Statement

Im ersten EXEC-Statement wird ein Programm aufgerufen, im zweiten und dritten jeweils eine Prozedur. Das Besondere hierbei ist, dass alle drei Formen gängig sind. Wird im EXEC-Statement ein Programm aufgerufen, dass die anstehende Verarbeitung durchführen soll, ist Angabe „PGM“ unerlässlich. Sobald die Angabe PGM weggelassen wird unterstellt das Job Entry Subsystem (JES) nämlich, dass hier eine Prozedur aufgerufen werden soll. In der Praxis führt dies zum Jobabbruch mit folgender Fehlermeldung:

```
PROCEDURE IEFBR14 WAS NOT FOUND
```

Abb. 4: Fehlermeldung bei Programmaufruf ohne PGM-Angabe

Ursache dafür ist, dass Programme und Prozeduren im Regelfall in verschiedenen Datasets gehalten werden. Programme werden in Program-Libraries, Prozeduren in Procedure-Libraries abgelegt. Wird also im EXEC-Statement ein Programm ohne PGM-Angabe aufgerufen, verzweigt das System auf die Procedure-Library und findet das Programm nicht.

Ein neues EXEC-Statement ist gleichzeitig der Beginn eines neuen Jobsteps, kurz Step genannt. Ein Job kann zwischen einem und 255 Steps enthalten. Sind mehr Steps enthalten, bricht der Job ab und gibt folgende Fehlermeldung aus:

```
EXCESSIVE NUMBER OF EXECUTE STATEMENTS
```

Abb. 5: Fehlermeldung bei mehr als 255 Steps

Der Ersteller eines Jobs sollte jedoch generell darauf achten, nicht zuviel Steps einzubauen, da der Job sonst sehr unübersichtlich wird.

2.3 Arbeitsmittel

Zu den einzelnen Arbeitsabläufen werden Arbeitsmittel in Form von DD-Statements (Data Definition) benötigt. Diese werden im Job direkt im Anschluss an die Spezifikation des entsprechenden Arbeitsablaufes definiert. Dahinter können weitere Arbeitsabläufe folgen.

Ein DD-Statement kann verschiedene Ausprägungen haben und von Programm zu Programm differieren. Es wird dazu benutzt werden, um dem ausgeführten Programm bzw. der ausgeführten Prozedur den Datenstrom bekannt zu geben. Bei einem einfachen File-Copy könnte dies folgendermaßen aussehen:

```
//FILECOPY EXEC PGM=IEBGENER
//SYSUT1   DD DSN=HLQ0.HLQ1.HLQ2,
//            DISP=SHR
//SYSUT2   DD DSN=HLQ10.HLQ11.HLQ12,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=*.SYSUT1,
//            SPACE=(TRK,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb. 6: File-Copy

In Abbildung 6 sind die DD-Statements

- SYSUT1
- SYSUT2
- SYSIN
- SYSPRINT

angegeben.

Je nachdem wie ein Programm aufgebaut ist kann es mindestens ein und maximal ∞ DD-Statements haben.

2.4 Der erste Job

Wir haben also in den Kapiteln 2.1 – 2.3 einzelne Job-Segmente kennengelernt. Wenn wir diese einzelnen Segmente zusammenfügen erhalten wir einen komplett lauffähigen Job, welchen den Inhalt einer Datei in eine andere Datei überspielt.

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//FILECOPY EXEC PGM=IEBGENER
//SYSUT1   DD DSN=HLQ0.HLQ1.HLQ2,
//            DISP=SHR
//SYSUT2   DD DSN=HLQ10.HLQ11.HLQ12,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=*.SYSUT1,
//            SPACE=(TRK,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb. 7: Der erste Job

Was bedeuten die Angaben in Abbildung 7 nun im Einzelnen?

- JOB00001

= Name des Job, unter dem er auf der Maschine ausgeführt wird. In den gängigen Output-Facilities (IOF, SDSF) kann der Joblauf durch diesen Namen herausgefiltert und der Joblauf beobachtet werden.

- JOB

= Angabe, die dem System einen Job ankündigt

- (123456)

= Accoutnummer, unter welcher der Joblauf abgerechnet wird

- SCHEIBE

= der Name des Job-Erstellers

- CLASS=A

= die Job-Klasse, unter welcher der Job ausgeführt werden soll. Die Auswahl der Job-Klasse richtet sich im allgemeinen nach der Job-Priorität und der Anzahl der Initiator, die unter der ausgewählten Job-Klasse verfügbar sind. Ein Initiator ist ein Systemprogramm mit eigenem Adress Space. Wann immer ein Initiator nichts zu tun hat fordert er das JES auf, ihm einen neuen Auftrag zuzuweisen. Der Initiator ist also der Teil des Systems, in dem der Job tatsächlich ausgeführt wird.

- MSGCLASS=T

= die Message-Klasse, über welche das Speichermedium bekannt gegeben wird, auf welches die Verarbeitungsprotokolle geschrieben werden sollen.

- NOTIFY=&SYSUID

= Bekanntgabe des Nachrichtenempfängers bei Jobende. Bei SYSUID handelt es sich um eine Systemvariable, die die User-ID des Job-Starters als Empfänger einsetzt. Dadurch wird es ermöglicht, einen Job von mehreren Benutzers starten zu lassen. Jeder Benutzer wird individuell über das Jobende informiert, ohne dass im NOTIFY die User-ID explizit angegeben werden muss.

- FILECOPY

= der Stepname. Dieser Name ist frei wählbar. Er darf allerdings nicht länger als acht Stellen sein. Um während des gesamten Joblaufes die einzelnen Stepinhalte besser nachvollziehen zu können ist es sinnvoll, einen Stepnamen zu wählen, der sich auf den Stepinhalt bezieht.

- EXEC

= Beginn eines Steps und Anweisung zur Programm- oder Prozedurausführung.

- PGM=IEBGENER

= Name des auszuführenden Programmes „IEBGENER“.

- SYSUT1

= über dieses DD-Statement erkennt das aufgerufene Programm, dass es sich um einen Eingabeeinheit handelt.

- SYSUT2

= über dieses DD-Statement erkennt das aufgerufene Programm, dass es sich um eine Ausgabeeinheit handelt.

- SYSIN

= Eingabe von Kommando-Parametern, die dem Programm genauere

Anweisungen geben, wie es die Daten zu verarbeiten hat.

- SYSPRINT

= im Programm definierte Angabe für die Erstellung der Verarbeitungsprotokolle. In dieser JCL bedeutet die Angabe „SYSOUT=*“, dass sich das Speichermedium, auf welches die Verarbeitungsprotokolle geschrieben werden sollen an der Message-Klasse orientiert.

- DD

= Data Definition; es gibt dem Betriebssystem Auskunft darüber, welche Ein- und Ausgabeeinheiten benötigt werden.

2.5 Umfeld der Job Control

Hier ist es wichtig zu wissen, welche Komponenten für die Ausführung eines Jobs notwendig sind.

- Das Betriebssystem führt Jobs durch
- Die JCL wird zur Spezifikation eines Jobs benötigt
- Das JES übersetzt die Job Control in Informationen für das Betriebssystem

Die Bearbeitung des Auftrages erfolgt durch das Betriebssystem. Der Dolmetscher, der die Kommunikation zwischen dem Anwender und dem Betriebssystem herstellt, ist das Job Entry Subsystem, kurz JES genannt. Es ist auch für den Start des Job Processing verantwortlich.

2.6 Job Entry Subsystem

Das JES hat die Aufgabe, Jobs anzunehmen und auf dem Rechner laufen zu lassen. Es steuert die Jobablaufkontrolle und leitet die Ergebnisse an die verschiedenen Ausgabeeinheiten weiter.

Es werden zwei JES-Varianten unterschieden:

Das JES2 und das JES3

Auf weitere Einzelheiten der beiden Systeme soll hier nicht weiter eingegangen werden. Nur soviel sei gesagt:

Obwohl beide Systeme ähnliche Funktionen haben, unterscheiden sie sich doch durch verschiedene Kontroll-Statements. JES2 ist von beiden Systemen das wesentlich weiter verbreitete.

3 JCL in der Praxis

3.1 Stepabfolge

Die Programmausführung innerhalb eines Steps wird häufig auch als Program Task bezeichnet.

Step Initiation
Program Task
Step Termination

Abb. 8: Stepabfolge

Bevor diese Zuordnung passieren kann, müssen die Datenträger und Datenbestände zunächst einmal reserviert werden. Dies geschieht in einer Vorlaufphase zum Programm, der Initiierung, auch Step Initiation genannt.

Ebenso müssen nach Ablauf eines Programms Datenbestände wieder freigegeben werden. Dies geschieht in der Step Termination. Es gibt zu jedem Programm jeweils eine Step Initiation und Step Termination, in die individuell Datenbestände reserviert und wieder freigegeben werden.

Noch einmal umgeben von einer Job Initiation und einer Job Termination, in denen Aktivitäten, die den gesamten Job betreffen, abgeleistet werden.

Job Initiation	
Step 1	
Step 2	**Program Task**
Step 3	
Job Termination	

Abb. 9: Stepabfolge innerhalb eines Jobs

Der in Abbildung 9 dargestellte Jobablauf wird in der Executionphase des JES aktiviert. Da die Datenbestände schon zugeordnet sein müssen, bevor das Programm anläuft, kann die Information, auf welche Datenbestände zugegriffen werden muss, nicht aus dem Programm gewonnen werden. Diese Aufgabe übernimmt zusätzlich noch die Job Control mit dem DD-Statement.

3.2 DSNAME

Über den DSNamen wird dem Betriebssystem mitgeteilt, welche Datenbestände für diesen Step reserviert werden sollen. Das Schlüsselwort des Parameters DSN oder DSName leitet sich aus dem Englischen von Datasetname ab.

Der Datasetname (DSN) kann aus mehreren Einzelteilen bestehen, den sogenannten Qualifiern bzw. High-Level-Qualifiern (HLQ).Diese dürfen aus maximal acht Stellen bestehen und werden durch Punkte voneinander getrennt. Der DSN darf inklusive Punkte maximal 44 Stellen lang sein.

In der Großrechnerwelt gibt es zwei unterschiedliche Dateiarten:

- Partitioned Sequentiell (PS-Datei)
- Partitioned Organisation (PO-Datei)

Die PS-Dateien erinnern eher an ein Verzeichnis unter Windows, nur dass hier anstelle des Backslash '\' ein Punkt '.' eingesetzt wird. Ein Dateiname könnte so aussehen:

- HLQ1.HLQ2.HLQ3

Diese Datei könnte direkt geöffnet und bearbeitet werden, wie die folgende Abbildung verdeutlicht:

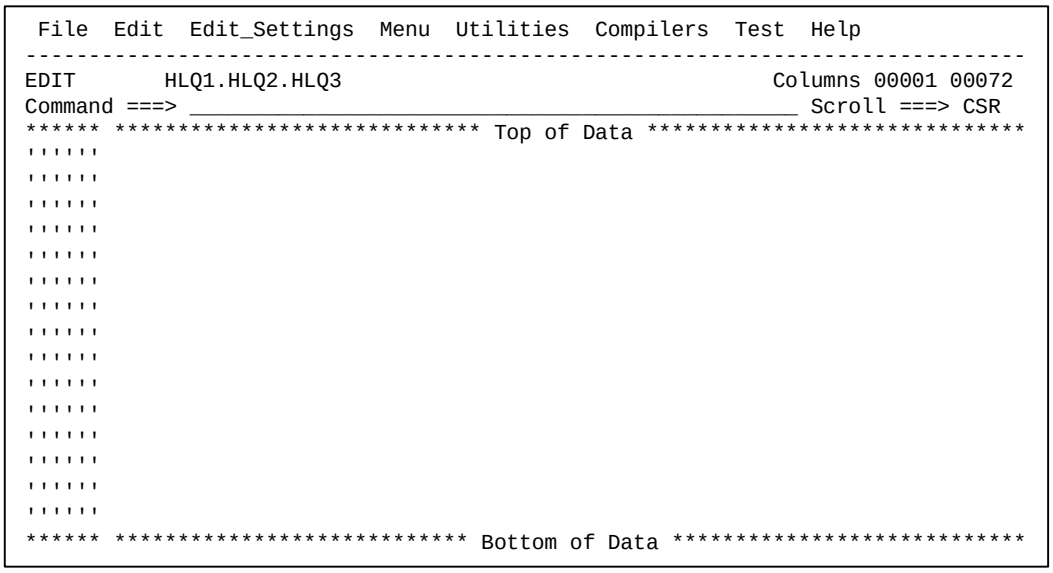

Abb. 10: Bearbeiten einer PS-Datei

In den PO-Dateien, den sog. „partitioned datasets“ werden, im Gegensatz zu PS-Dateien, also den rein sequentiellen Datenbeständen, die Dateieinträge in Membern zusammengefasst. Die Namen dieser Member dürfen maximal acht Stellen lang sein[4]. Beim Öffnen einer PO-Datei werden alle sich darin befindlichen Member aufgelistet.

[4] siehe Kapitel 8.3.3 DSORG – Dataset-Organisation

```
 Menu  Functions  Confirm  Utilities  Help
------------------------------------------------------------------------------
EDIT              HLQ1.HLQ2.HLQ3                          Row 00032 of 00056
Command ===>_____________________________________________ Scroll ===> CSR
           Name     Prompt       Size   Created          Changed          ID
_________ JOB00001 *Edited          9  2003/07/18  2007/02/23 13:19:08  USER1
_________ JOB00002                 48  2003/07/31  2004/08/04 14:59:59  USER1
_________ JOB00003                 88  2004/03/30  2006/08/10 10:49:33  USER1
_________ JOB00004                162  2003/07/16  2006/08/10 10:09:18  USER1
_________ JOB00005                 40  2003/08/19  2005/01/10 14:40:56  USER1
_________ JOB00006                 11  2004/03/31  2006/07/03 15:33:37  USER1
_________ JOB00007                 11  2004/03/31  2005/11/02 12:48:53  USER1
_________ JOB00008                 20  2004/04/06  2004/04/08 08:52:20  USER1
_________ JOB00009                 30  2004/04/07  2006/01/12 14:46:05  USER1
_________ JOB00010                 17  2005/02/16  2006/02/20 11:03:27  USER1
_________ JOB00011                 54  2004/07/02  2005/02/14 10:58:25  USER1
_________ JOB00012                 57  2004/07/29  2004/07/29 10:03:52  USER1
_________ JOB00013                 60  2004/08/30  2004/09/20 14:27:21  USER1
_________ JOB00014                 18  2004/08/31  2004/08/31 09:45:11  USER1
_________ JOB00015                 51  2004/09/20  2007/01/16 09:26:57  USER1
_________ JOB00016                 44  2005/02/04  2006/09/07 13:17:27  USER1
_________ JOB00017                  7  2005/02/15  2005/02/15 09:53:48  USER1
_________ JOB00018                 14  2005/04/06  2005/11/16 11:38:41  USER1
```

Abb. 11: Anzeige einer PO-Datei

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       HLQ1.HLQ2.HLQ3(MEMBER01) - 01.99               Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
****** **************************** Bottom of Data ****************************
```

Abb. 12: Bearbeiten eines Members

Wie aus den Abbildungen 11 und 12 gut zu erkennen ist, wird bei der Bearbeitung eines Members der Membername 'MEMBER01' in Klammern hinter den Dateinamen 'HLQ1.HLQ2.HLQ3' angefügt.

Bei der Abwicklung von Jobs kommt es immer wieder vor, dass in ihrem Verlauf für kurze Zeit Zwischendatenbestände benötigt werden, z.B. als Hilfsdatenbestände. Da diese Datenbestände nur innerhalb des Jobs benötigt werden, kann man sie als temporäre Datenbestände anlegen. Dazu ist eine Kennzeichnung zur Differenzierung von anderen Datenbeständen notwendig.

Die Namen für temporäre Datenbestände beginnen mit

- zwei Ampersands[5] (&&).

Neben den frei zu vergebenen DSNamen gibt es auch einen Namen, der in der Job Control fest vorgegeben ist. Dies ist der DSName

- NULLFILE.

Dieser Parameter hat genau die gleiche Wirkung wie die Angabe des positionellen Parameters DUMMY.

Über den DSNamen wurde dem Betriebssystem mitgeteilt, mit welchem Datenbestand gearbeitet werden soll. Als zusätzlichen Hinweis benötigt das Betriebssystem nun einen Hinweis, ob dieser Datenbestand schon existiert oder ob er neu angelegt werden soll. Darüber hinaus muss dem Betriebssystem bekannt gegeben werden, was mit dem Datenbestand nach Programmende zu tun ist. Beide Informationen werden über den Parameter

- DISP

bekannt gegeben.

Die folgenden Beispiele zeigen, wie sequentielle Dateien bzw. Member in PO-Dateien angesprochen werden können:

```
…
//EINGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=SHR
…
```

Abb. 13: Ansprechen einer sequentiellen Datei

Unter dem DD-Namen “Eingabe1” in Abbildung 13 wird ein sequentieller Datenbestand mit dem Namen “HLQ1.HLQ2.HLQ3” angesprochen. Die Punkte in der zweiten Zeile besagen, dass hier in jedem Falle noch etwas kommen muss.

[5] vgl. Kapitel 3.13

```
…
//EINGABE1 DD DSN=HLQ10.HLQ11.HLQ12(MEMBER1),
//            DISP=SHR
…
```

Abb. 14: Ansprechen eines Members

Unter dem DD-Namen „Eingabe1“ in Abbildung 14 wird das Member „MEMBER1“ Dataset „HLQ10.HLQ11.HLQ12“ angesprochen. Es kann sich bei diesem Dataset um eine Library oder um eine PO-Datei handeln. Das Member wird vom Betriebssystem so behandelt wie ein sequentieller Datenbestand.

```
…
//EINGABE1 DD DSN=&&TMPDSN,
//            DISP=…
```

Abb. 15: Ansprechen eines temporären Datenbestandes

In Abbildung 15 wird ein temporärer Datenbestand unter dem logischen Namen „TMPDSN“ benutzt. Das System definiert sich selbst einen vollständigen Namen daraus. Auch hier gilt, dass noch weitere Parameter gebraucht werden.

```
…
//EINGABE1 DD DSN=NULLFILE
…
```

Abb. 16: Ansprechen eines NULLFILES

Die Vorgehensweise durch das Betriebssystem ist bei der Angabe des Parameters „DSN=NULLFILE“ in Abbildung 16 identisch mit der Vorgehensweise bei der Angabe des positionellen Parameters DUMMY.

3.3 DISP

Die folgende Abbildung zeigt den allgemeinen Aufbau des DISP-Parameters:

```
{DISP=status                                                              }
{DISP=([status][,normal-termination-disp][,abnormal-termination-disp])}
```

Abb. 17: Allgemeiner Aufbau des DISP-Parameters

’status’ beschreibt den Zustand des Datenbestandes vor seiner Verwendung. Er trifft eine Aussage darüber, ob der Datenbestand neu anzulegen ist oder ob auf einen existierenden Datenbestand zurückgegriffen werden soll, d.h. status ist ausschließlich in der Step-Initiation interessant.

’normal-termination-disp’ gibt an, was mit dem Datenbestand geschehen soll, wenn der Step zu einem normalen Ende kommt. Es geht hier um die Entscheidung, ob und in welcher Form der Datenbestand erhalten bleiben soll. Dieser Parameter wirkt in der Step-Termination.

’abnormal-termination-disp’ gibt an, was mit dem Datenbestand geschehen soll, wenn der Step zu einem abnormalen Ende kommt. Dieser Parameter ersetzt den zweiten Parameter im Falle eines Programmabbruches in diesem Step. Ansonsten ist dieser Parameter bedeutungslos.

Die folgende Tabelle zeigt, welche Werte der Subparameter ‚status' annehmen kann:

	Status	normal termination	abnormal termination
	NEW	,DELETE	,DELETE
	OLD	,KEEP	,KEEP
DISP=	SHR	,PASS	,CATLG
	MOD	,CATLG	,UNCATLG
	,	,UNCATLG	
		,	

Tab. 1: Mögliche Werte des Subparameters 'status'

3.3.1 Status

3.3.1.1 NEW

Der Datenbestand existiert noch nicht und soll in der Stepinitiation dieses Steps neu angelegt werden.

3.3.1.2 OLD

Der Datenbestand existiert, allerdings benötigt diese Anwendung den Datenbestand exklusiv. Kein anderer Job und keine andere Anwendung darf gleichzeitig auf ihn zugreifen. Bei schreibendem Zugriff wird zunächst die Endeadresse des Datenbestandes auf Null gesetzt und somit der Datenbestand von Anfang an wieder neu geschrieben.

3.3.1.3 SHR

Der Datenbestand existiert. Es darf von mehreren Jobs gleichzeitig auf diesen Datenbestand zugegriffen werden. Man sagt, der Datenbestand wird zwischen verschiedenen Anwendern „geshared". Das Betriebssystem geht in jedem Fall davon aus, dass es auf einen vorhandenen Datenbestand zugreifen kann.

3.3.1.4 MOD

Dieser Wert ist prinzipiell mit dem Wert OLD vergleichbar. Es gibt allerdings einige gravierende Unterschiede. Im Gegensatz zum Wert OLD wird die Endeadresse bei schreibendem Zugriff nicht auf Null gesetzt, sondern sie bleibt erhalten, d.h. die Datensätze, die geschrieben werden, werden der Zieldatei angefügt.

3.3.2 Normal Termination

3.3.2.1 DELETE

Der Datenbestand wird bei Stepende gelöscht. Falls der Datenbestand katalogisiert war und über den Katalog auf den Datenbestand zugegriffen worden ist, wird auch der Katalogeintrag gelöscht.

3.3.2.2 KEEP

Der Datenbestand bleibt über das Ende des Steps hinaus erhalten, wird allerdings, falls er neu ist, nicht katalogisiert.

3.3.2.3 PASS

Der Datenbestand bleibt über das Ende hinaus erhalten. Der Programmierer trifft an dieser Stelle keine endgültige Entscheidung über die Disposition. Wenn der Programmierer in einem späteren Step keine endgültige Entscheidung trifft, wird bei Jobende die Standardeinstellung zum Zuge kommen. Diese ist bei neuen Datenbeständen DELETE.

3.3.2.4 CATLG

Der Datenbestand bleibt auch über das Ende des Steps hinaus erhalten und wird in einem Katalog eingetragen. Dieser Parameter sollte beim Anlegen neuer Datenbestände gewählt werden, da auf diese Datenbestände über den Katalog zugegriffen werden kann. Dabei ist es völlig egal, auf welchem Datenträger der katalogisierte Datenbestand gespeichert ist.

3.3.2.5 UNCATLG

Der Datenbestand bleibt erhalten, allerdings wird ein vorhandener Katalogeintrag gelöscht. Dieser Parameter ist eigentlich nur für Ausnahmesituationen gedacht, da Datenbestände nach Möglichkeit heute alle katalogisiert sein sollten.

3.3.3 Abnormal Termination

Die Angaben zur „abnormal termination“ sind praktisch identisch mit denen zur „normal termination“.

3.3.4 Standardeinstellungen (defaults)

Die folgende Tabelle zeigt die Standardeinstellung der Subparameter 'status' und 'normal-termination':

status	normal-termination-disp
NEW	DELETE
nicht NEW	KEEP

Tab. 2: Standardeinstellungen

Wie bereits erläutert wird 'NEW' als Standardeinstellung genommen, wenn keine angaben gemacht werden.

Für die 'normal-termination-disp' wird der Default abhängig vom 'status' des Datenbestandes bestimmt: Das Betriebssystem versucht hier den Zustand, wie er vor Beginn des Steps war, wiederherzustellen.

Für die 'abnormal-termination-disp' übernimmt das Betriebssystem den Subparameter der normalen Termination.

Bei der Ermittlung des abhängigen Defaults ist es unerheblich, ob die bestimmenden Parameter explizit angegeben wurden oder ob sie selbst Defaults sind. Da es nicht immer ganz einfach ist, die Defaultwerte zu bestimmen, sollte generell gelten:

- Die Disposition sollte immer voll qualifiziert angegeben werden!

Fazit: Mit den Parametern DSN und DISP kann jeder existierende und katalogisierte Datenbestand angesprochen werden. Diese Angaben reichen völlig aus. Mehr sollten bei einem existierenden Datenbestand auch nicht angegeben werden.

Das folgende Beispiel zeigt, wie der Subparameter DISP zum Einsatz kommt:

```
…
//EINGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=SHR
…
```

Abb. 18: Erste Einsatzmöglichkeit des DISP-Subparameters

Durch 'DISP=SHR' wird auf einen existierenden Datenbestand zugegriffen. Gleichzeitig dürfen auch noch andere Anwendungen auf den Datenbestand zugreifen.

```
…
//EINGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=OLD
…
```

Abb. 19: Zweite Einsatzmöglichkeit des DISP-Subparameters

Der Datenbestand in Abbildung 19 ist exklusiv für diesen Step reserviert. Andere Anwendungen, die diesen Datenbestand bearbeiten wollen, müssen mit ihrer Arbeit warten, bis der Step zu Ende ist.

```
…
//EINGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            …
```

Abb. 20: Dritte Einsatzmöglichkeit des DISP-Subparameters

Der Datenbestand in Abbildung 20 soll neu angelegt und katalogisiert werden. Im Falle eines Abbruches wird der Datenbestand gelöscht.

```
…
//EINGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(OLD,DELETE,KEEP),
…
```

Abb. 21: Vierte Einsatzmöglichkeit des DISP-Subparameters

Der Datenbestand in Abbildung 21 existiert und wird diesem Step exklusiv zugeordnet. In der Step Termination wird der Datenbestand gelöscht und entkatalogisiert. Im Falle eines „abnormal ends“, also bei einem Jobabbruch, soll der Datenbestand in seiner jetzigen Form erhalten bleiben.

Hinweis: Als 'abnormal termination' an dieser Stelle CATLG einzusetzen wäre falsch, denn der Datenbestand sollte bereits katalogisiert sein. Das Betriebssystem würde in diesem Fall eine Warnung verfassen und normal weiterarbeiten.

```
…
//EINGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,PASS),
//            …
```

Abb. 22: Fünfte Einsatzmöglichkeit des DISP-Subparameters

Der Datenbestand in Abbildung 22 wird neu angelegt. Die Entscheidung über die Dispositionen wird auf später verschoben. Wenn in keinem späteren Step dieses Jobs eine andere Disposition entschieden wird, wird dieser Datenbestand bei Jobende gelöscht.

3.4 DCB

Für jeden Datenbestand muss ein DCB angelegt werden. Das Anlegen kann durch das Programm oder mit Hilfe der Job Control erfolgen. Der DCB muss im Programm nicht komplett angegeben werden. Das Programm kennt ja nur den logischen Aufbau des Datenbestandes, weiß aber nichts von den Eigenheiten des physischen Datenträgers, auf dem der Datenbestand gespeichert wird.

Die Schnelligkeit der Verarbeitung und die Speicherplatzausnutzung für sequentielle Datenbestände hängen ganz entscheidend von der Auswahl einer optimalen und damit richtigen Blocklänge ab. Die optimale Blocklänge hängt wesentlich von dem Einheitentyp ab, auf dem der Datenbestand abgelegt wird.

Allerdings ist in der Regel der Einheitentyp, auf dem der Datenbestand abgelegt werden soll, zur Zeit der Kodierung des Programms noch gar nicht bekannt. Aus diesem Grund gibt es in der Job Control Parameter, mit denen die DCB-Informationen vervollständigt werden können. Durch diese Angaben können die DCB-Informationen immer an die aktuelle Situation angepasst und die Verarbeitung somit optimiert werden.

Der Parameter DCB kann allerdings die DCB-Festlegungen, die im Programm definiert wurden, nicht überschreiben. Wenn also flexibel gearbeitet werden

soll, achte man darauf, im Programm nur das unbedingt notwendige festzulegen. Alle anderen Angaben sollten beim Neuanlegen eines Datenbestandes in der Job Control festgelegt werden. Die wichtigsten Angaben sind:

Parameter	Beschreibung
LRECL	Definition der logischen Satzlänge; überreicht dem Betriebssystem die Information über die Länge der Datensätze, wie sie zwischen Puffer und Programm ausgetauscht werden.
BLKSIZE	Definition der physischen Blöcke; mit dem Parameter BLKSIZE wird dem Betriebssystem die Information über die Länge der Blöcke überreicht, wie sie zwischen Puffer und externem Datenträger ausgetauscht werden.
RECFM	Definition der Satzstruktur; mit dem Parameter RECFM wird das Format und die Charakteristik der Datensätze spezifiziert.
DCB	Wahlweise können die Parameter LRECL, BLKSIZE und RECFM auch als Subparameter im DCB kodiert werden.

Tab. 3: Die wichtigsten Parameterangaben für den DCB

Die wichtigsten Rekordformate (RECFM) sind:

Format	Bedeutung
FB	Feste Satzlänge, Blocklänge ist ein ganzhaliges Vielfaches der Satzlänge, z.B. Satzlänge x Multiplikator = Blocklänge 80 x 1 = 80 80 x 2 = 160 80 x 3 = 240 … 80 x 30 = 2400
FBA	Entspricht FB; das erste Datenbyte enthält ein ISD/ANSI-Steuerzeichen. Es wird in Verbindung mit dem Drucken von Daten verwendet, wird aber heute so gut wie nicht mehr benötigt.
VB	Variable Satzlänge; vom Prinzip her wie „FB“, nur dass jeder Datensatz ab Stelle fünf beginnt und eine individuelle Satzlänge hat. Die ersten vier Stellen enthalten Informationen zum Datensatz.
VBA	Entspricht VB; das erste Datenbyte enthält ein ISD/ANSI-Steuerzeichen. Es wird in Verbindung mit dem Drucken von Daten verwendet, wird aber heute so gut wie nicht mehr benötigt. Die Datensätze werden nur im EDIT-Modus komplett angezeigt, im BROWSE-Modus ist die erste Stelle nicht zu sehen.
U	Records undefinierter Länge. LRECL ist wegzulassen. BLKSIZE muss > oder = der Länge des längsten Records des Datasets sein, jedoch mindestens 80 Stellen lang sein. Eine Datei mit dieser Satzlängendefinition kann nicht editiert werden.

Tab. 4: Die wichtigsten Rekordformate

Die Angaben 'LRECL und BLKSIZE' spezifizieren die Satzlänge und die Anzahl der Datensätze, die zu einem Block zusammengefasst werden, nach folgender Berechnungformel:

Blockgröße / Logische Satzlänge = Anzahl Sätze
800 / 80 = 10

```
…
//AUSGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=3380,
//            RECFM=FB,
//            LRECL=80,
//            BLKSIZE=800,
//            SPACE=(80,(1,10000),RLSE)
…
//AUSGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=3380,
//            LRECL=80,
//            DCB=(LRECL=80,BLKSIZE=800),
//            SPACE=(80,(1,10000),RLSE)
…
```

Abb. 23: Angabe von LRECL und BLKSIZE

Die Angaben können sowohl einzeln angegeben werden als auch unter dem Parameter 'DCB' zusammengefasst werden. Abbildung 23 zeigt die sowohl die Möglichkeit der Einzelangabe als auch die Zusammenfassung unter DCB.

Abbildung 23 zeigt ein Beispiel, in dem der Spoolausgabebestand eine Satzlänge von 131 Bytes hat und zu jeweils 20 Sätzen geblockt wird. Folgende Berechnungsformel wird zugrunde gelegt:

Blockgröße / Logische Satzlänge = Anzahl Sätze
2620 / 131 = 20

```
…
//AUSGABE1 DD SYSOUT=A,,
//            RECFM=FBA,
//            LRECL=131,
//            BLKSIZE=2620,
…
```

Abb. 24: Angabe von LRECL und BLKSIZE unter SYSOUT

Das erste Byte eines jeden Datensatzes benutzt der Drucker zur Vorschubsteuerung.

3.5 SPACE

Speicherplatz auf einem ausgewählten Datenträger zu reservieren ist die Aufgabe des SPACE-Parameters. Das folgende Beispiel zeigt die allgemeine Syntax des SPACE-Parameters:

```
SPACE=({TRK,    }(primary-qty[,second-qty][,directory]))
SPACE=({TRK,    }(primary-qty[,          ][,index    ]))
SPACE=({CYL,    }(primary-qty[,second-qty][,directory]))
SPACE=({CYL,    }(primary-qty[,          ][,index    ]))
SPACE=({blklgth,}(primary-qty[,second-qty][,directory]))
SPACE=({blklgth,}(primary-qty[,          ][,index    ]))
SPACE=({reclgth,}(primary-qty[,second-qty][,directory]))
SPACE=({reclgth,}(primary-qty[,          ][,index    ]))
```

Abb. 25: Allgemeiner SPACE-Parameter

Der erste Parameter in Abbildung 25 dient dazu, den Maßstab anzugeben, mit dem die nachfolgenden Werte zu interpretieren sind. Maßstab bedeutet hier, ob die Speicherplatzreservierung in

- Zylinder (CYL),
- Spuren (TRK) oder
- Bytes (blklgth, reclgth)

angegeben werden soll.

Mit 'primary-qty' wird die Anzahl der Einheiten, die reserviert werden soll, spezifiziert. Die hier definierten Mengeneinheiten werden stets auf einem einzigen Volume angelegt.

Mit der 'second-qty' wird die Anzahl der Einheiten spezifiziert, die reserviert werden soll, falls der in der 'primary-qty' reservierte Platz nicht ausreicht. Die 'second-qty' hat 15 Extents.

Die folgende Tabelle erleutert die Berechnung des belegten Speicherplatzes. Es ist jedoch zu berücksichtigen, dass es sich hier um eine Berechnung für den Idealfall handelt und die Angaben in der Praxis differieren können.

SPACE-Angabe	Maßstab	Max. Speicherbelegung
SPACE=(TRK(**1,10000**),)	TRK = 56.000 Byte	56000 x 1 + 10.000 x 15 = 56.000 x 150.000 = 8.400.000.000 Byte
SPACE=(80,(**2,1000**),)	80 Byte	80 x 2 + 1.000 x 15 = 160 x 15.000 = 240.000 Byte
SPACE=(CYL,(**10,10**),)	CYL = 15 TRKS	15 x 56.000 x 10 + 10 x 15 x 15 x 56.000 = 8.400.000 + 126.000.000 = 134.400.000 Byte

Tab. 5: Max. Speicherbelegung durch primary- und secondary-qty

Werden neue Datenbestände angelegt, stehen sie in der Regel wild verstreut über dem Datenträger. Das Betriebssystem holt sich den Platz daher, wo es ihn am schnellsten findet. Der Zusammenhalt der einzelnen Datenbestände wird über das Inhaltsverzeichnis des Datenträgers, den

- VTOC (Volume Table of Contents)

gewährleistet.

Soll ein Partitioned Dataset angelegt werden, auch Bibliothek genannt, wird innerhalb des Datenbestandes noch ein Inhaltsverzeichnis benötigt. Dieses Inhaltsverzeichnis wird benötigt, um die einzelnen Mitglieder der Bibliothek wiederzufinden.

Das Inhaltsverzeichnis besteht aus 256 Byte langen Blöcken. Diese Blockgröße des Inhaltsverzeichnisses ist unabhängig von der Struktur des Datenbestandes selbst. Ein solcher Block ist für die Adressierung von ca. fünf Membern ausreichend.

SPACE-Angabe	Anzahl Member (ca. Angabe)
SPACE=(TRK,(1,10000,**10**),)	10 x 5 = 15 Member
SPACE=(80,(1,10000,**50**),)	50 x 5 = 250 Member
SPACE=(CYL,(1,10000,**200**),)	200 x 5 = 1000 Member

Tab. 6: Berechnung der Member-Quantität

Wie aus Tabelle 6 (vgl. Tabelle 5) hervorgeht, wird mit dem 'directory'-Eintrag die Anzahl der 256-Byte-Blöcke angegeben, die als Directory reserviert werden sollen. Fehlt der 'directory'-Eintrag in der Job Control, legt das Betriebssystem einen sequentiellen Datenbestand an.

Die Verarbeitung der Daten erfolgt immer satzweise. Dazu wird jeweils immer ein Datensatz aus dem Puffer in den Programmbereich übertragen und zurückgeschrieben. Man spricht bei den Datensätzen auch von 'logical records'. 'Ein Datensatz = Logical Record' ist die Übertragungseinheit zwischen Puffer und Programm.

PGM
OPEN FILE HUGO
READ FILE HUGO
WRITE FILE HUGO
Logical Record
Logical Record

Abb. 26: Datensatz als Übertragungseinheit

Die im Puffer befindlichen Datensätze werden in einer Einheit auf einen externen Datenträger übertragen. Man spricht bei dieser Übertragungseinheit auch von einem Block.

- Ein Block ist die Übertragungseinheit zwischen Puffer und physischem Datenträger.

```
…
//AUSGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=3380,
//            RECFM=FB,
//            LRECL=80,
//            BLKSIZE=800,
//            SPACE=(80,(1,10000),RLSE)
…
```

Abb. 27: Angabe des SPACE-Parameters mit 80er-Stelleneinheiten

Mit der SPACE-Angabe im Beispiel aus Abbildung 27 wird eine 80er-Stelleneinheit sofort reserviert. Es können also 80 Zeichen (= 80 Byte) sofort abgelegt werden. Wenn diese 80 Byte, die sog. 'primary-qty' nicht ausreicht, versucht das Betriebssystem über die sog. 'secondary-qty' zusätzlichen Speicher zu reservieren. Dabei werden bis zu 15 mal jeweils 10000 Einheiten in der Größenordnung von 80 Byte dazu geholt. Da keine Directory-Angabe gemacht wurde handelt es sich um einen sequentiellen Datenbestand. Die Angabe 'RLSE' (=release) veranlasst, dass nicht benötigter Speicherplatz wieder freigegeben wird.

```
…
//AUSGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=3380,
//            RECFM=FB,
//            LRECL=80,
//            BLKSIZE=800,
//            SPACE=(TRK,(1,10000,10),RLSE)
…
```

Abb. 28: Angabe des SPACE-Parameters mit Directory-Angabe

Um das Beispiel in Abbildung 28 zu erläutern und logische Unklarheiten zu beseitigen, ist eine Umrechnung erforderlich. Aus Plattensicht gibt es zwei Speichergrößen: Tracks (TRK [=Spuren]) und Zylinder (CYL). Ein Track hat eine Speicherkapazität von 56.000 Bytes und stellt die kleinste adressierbare Einheit auf einer Platte dar.

Die Anzahl der Tracks innerhalb eines Zylinders hängt vom Plattentyp, d.h. aus wie vielen Schichten eine Platte besteht. Besteht eine Platte beispielsweise aus 15 Schichten, hat ein Zylinder 15 Tracks, besteht sie dagegen aus 20 Schichten, hat ein Zylinder 20 Tracks.

In der SPACE-Angabe in Abbildung 28 werden also nicht, wie in den vorangegangenen Beispielen, 80 Bytes, sondern 56.000 Bytes sofort reserviert. Von diesen 56.000 Bytes werden 10 mal 256 Byte für das Directory reserviert.

Durch die Angabe des dritten Subparameters in der inneren Klammer erkennt das Betriebssystem, dass es sich um einen Partitioned Data Set handelt. Die 10 gibt die Anzahl der zu reservierenden 256 Byte großen Directory-Blöcke an.

3.6 UNIT

Wenn ein Datenbestand neu angelegt werden soll, muss dem Betriebssystem explizit mitgeteilt werden, auf welchem Einheitentyp dieser Datenbestand abgelegt werden soll. Hierfür gibt es verschiedene Möglichkeiten:

1. Physische Adresse (wird selten gewählt)
   Die physische Adresse spezifiziert eine einzelne Einheit. Welche Einheiten unter welcher Adresse ansprechbar sind, kann sich schnell ändern. Diese Form der Angabe ist eigentlich nur etwas für einige, wenige ausgewählte Anwendungen.

2. Spezifizierung des Einheitentyps
   Mit dem Einheitentyp wird eine ganze Gruppe von Datenträgern des gleichen Typs angesprochen. Allerdings können in einer Installation die Datenträgertypen wechseln.

3. Name von Einheitengruppen
   Das ist der Name, unter dem eine Gruppe von Einheiten zusammengefasst ist. In dieser Form wird ein logischer Name angesprochen, unter dem verschiedene Einheiten zusammengefasst sein können. Das diese Zuordnung schnell zu ändern sind, ist dies die ideale Möglichkeit der Datenträgeranforderung.

Über die UNIT-Angabe wird der entsprechende Einheitentyp spezifiziert, z.B. eine Bandeinheit oder ein bestimmter Plattentyp, auf dem ein Datenbestand angelegt oder gesucht werden soll.

Das folgende Beispiel zeigt den allgemeinen Aufbau der UNIT-Syntax:

```
UNIT=([device-number][,unit-count][,DEFER])
     ([device-type  ][,P          ][,DEFER])
     ([group-name   ][,           ][,DEFER])
```

Abb. 29: Die allgemeine UNIT-Syntax

Mit der UNIT-Angabe wurde spezifiziert, auf welchem Einheitentyp der Datenbestand abgelegt werden soll. Je nach Schreibweise wurde unter Umständen auch eine ganz bestimmte Einheit spezifiziert.

Wenn jedoch an Band- oder Kassetteneinheiten gedacht wurde, besteht hier zusätzlich die Möglichkeit, auf dieser Einheit einen entsprechenden Datenträger zu montieren.

Für Platteneinheiten bestehen prinzipiell die gleichen Möglichkeiten, allerdings ist bei modernen Platten das Auswechseln selten. Hier wird dieser Parameter benutzt, um aus einer UNIT-Gruppe einen einzelnen Datenträger für einen Datenbestand auszuwählen.

Das folgende Beispiel zeigt die Anwendung des UNIT-Parameters in der Job Control:

```
…
//AUSGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=3380,
//            RECFM=FB,
//            LRECL=80,
//            BLKSIZE=800,
//            SPACE=(80,(1,10000),RLSE)
…
```

Abb. 30: Anlage eines Datenbestandes auf einer 3380-UNIT

Der Datenbestand in Abbildung 30 wird auf einer Platteneinheit vom Typ 3380 angelegt.

```
…
//AUSGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=TEST,
//            RECFM=FB,
//            LRECL=80,
//            BLKSIZE=800,
//            SPACE=(80,(1,10000),RLSE)
…
```

Abb. 31: Anlage eines Datenbestandes auf einer TEST-UNIT

Der Datenbestand in Abbildung 31 wird auf einer Platteneinheit angelegt, welche der UNIT-Gruppe TEST zugeordnet ist.

Voraussetzung für die Realisierung der in den Abbildungen 30 und 31 dargestellten Beispiele ist natürlich, dass die entsprechenden Platteneinheiten und UNIT-Gruppen installiert sind. Welche UNIT-Gruppe existieren bzw. mit welcher gearbeitet werden darf ist von der Installation abhängig.

```
…
//AUSGABE1 DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=3380,
//            VOL=SER=USER05,
//            RECFM=FB,
//            LRECL=80,
//            BLKSIZE=800,
//            SPACE=(80,(1,10000),RLSE)
…
```

Abb. 32: Anlage eines Datenbestandes auf einer 3380-UNIT mit Datenträgernamen

Der Datenbestand in Abbildung 32 wird auf einer Platteneinheit vom Typ 3380 mit dem Datenträgernamen USER05 angelegt. Voraussetzung hierfür ist selbstverständlich, dass in der Installation ein Datenträger mit dem Namen USER05 auf einer Einheit vom Typ 3380 installiert ist. Im anderen Fall wird der Operator vom Betriebssystem aufgefordert, den entsprechenden Datenträger zu montieren.

3.6.1 device-number

Der 'device-number'-Parameter spricht eine bestimmte, physikalische Einheit an. Diese Angabe sollte nur benutzt werden, wenn es unbedingt nötig ist.

3.6.2 device-type

Der 'device-type'-Parameter gibt einen Einheitentyp an, der verwendet werden soll. Der Einheitentyp ist ein von der IBM definierter Name für einen bestimmten Maschinentyp, z.B. 3380 oder 3390 für ein bestimmtes Plattenmodell.

3.6.3 group-name

Der 'group-name'-Parameter ist der symbolische Name für eine Gruppe von Geräten, der bei der Installation festgelegt wird. Der 'group-name' wird immer dann verwendet, wenn der Datenbestand nicht auf bestimmten Einheitentypen gespeichert werden soll. Die Zuordnung erfolgt hier auf einen Einheitentyp, der dieser Gruppe zugeordnet ist

3.6.4 unit-count

Der 'unit-count'-Parameter spezifiziert die Anzahl von Einheiten, die diesem Datenbestand zugeordnet werden sollen.

3.6.5 P

Der 'P'-Parameter steht für parallel mountig und bedeutet, dass mehrere Bänder zur gleichen Zeit aufzulegen sind.

3.6.6 DEFER

Der 'DEFER'-Parameter gibt an, dass der Operator die Anweisung zum Einlegen des Bandes erst erhält, wenn der Datenbestand eröffnet wird. Anderherum: Wenn der Datenbestand nicht eröffnet wird, braucht der Operator das Band gar nicht erst zu montieren.

- Die Subparameter 'P' und 'DEFER' haben ausschließlich Gültigkeit für die Bandverarbeitung.

3.7 VOLUME

Dieser Parameter ist der VOLUME-Parameter. Die folgenden Beispiele zeigen die allgemeine Syntax des VOLUME-Parameters:

```
{VOLUME}=(…[,][SER=serial-number                    ])
{VOLUME}=(,[,][SER=(serial-number[,serial-number]…)])
{VOLUME}=(,[,][REF=dsname                           ])
{VOLUME}=(,[,][REF=*.ddname                         ])
{VOLUME}=(,[,][REF=*.stepname.ddname                ])
{VOLUME}=(,[,][REF=*.stepname.procstepname.ddname   ])
```

Abb. 33: Allgemeiner VOLUME-Parameter

Eine alternative Schreibweise des VOLUME-Parameters wird in folgendem Beispiel dargestellt:

```
{VOL}=(…[,][SER=serial-number                    ])
{VOL}=(,[,][SER=(serial-number[,serial-number]…)])
{VOL}=(,[,][REF=dsname                           ])
{VOL}=(,[,][REF=*.ddname                         ])
{VOL}=(,[,][REF=*.stepname.ddname                ])
{VOL}=(,[,][REF=*.stepname.procstepname.ddname   ])
```

Abb. 34: Allgemeiner VOLUME-Parameter (alternative Schreibweise)

Die Subparameter 'SER' und 'REF' sind, wie VOLUME oder VOL selbst, Schlüsselwortparameter. Mit 'SER' wird eine oder mehrere Datenträgernummern angegeben. Mehrere Nummern werden benötigt, wenn ein Datenträger nicht ausreicht, um die Datenbestände aufzunehmen. Als 'serial-number' können nur die Nummern benutzt werden, die beim Aufbau des Datenträgerkataloges vergeben wurden und das ist Sache der Datenadministration.

'REF' stellt einen Rückbezug her. 'REF=dsname' bezieht sich auf den Namen eines katalogisierten Datenbestandes. 'REF=ddname' verweist auf einen DD-Namen, der vorher im gleichen Step erklärt wurde.

Wird noch zusätzlich der Name des Steps angegeben, besteht auch die Möglichkeit, sich auf DD Statements aus anderen Steps zu beziehen. Voraussetzung ist allerdings auch hier, dass diese DD Statements vor dem Rückbezug liegen.

3.8 Spezielle DD-Statements

In jeder Installation gibt es DD-Namen, die nicht spezifiziert werden müssen. Dazu gehören u.a. Standardbibliotheken sowie der DD-Namen SYSIN. Wird eine andere Verarbeitung als die in der Installation festgelegte gewünscht, können eigene DD-Statements definiert werden.

Die Programme, die in Jobs aufgerufen werden, werden in einer Bibliothek gesucht, deren Namen der Programmierer normalerweise nicht zu sehen bekommt. Das ist in der Regel eine Standardbibliothek. Die folgenden Beispiele zeigen Job Control, in der nicht auf eine Standardbibliothek, sondern auf eine eigene Bibliothek zugegriffen wird:

```
…
//JOBLIB   DD DSN=HLQ1.HLQ2,
//            DISP=SHR
//         DD DSN=HLQ3.HLQ4.HLQ5,
//            DISP=SHR
//         DD DSN=HLQ6.HLQ7,
//            DISP=SHR
…
```

Abb. 35: Joblib-Angabe

Soll während der Verarbeitung eines Jobs auf eine andere Bibliothek zurückgegriffen werden, so muss ein 'JOBLIB DD-Statement' kodiert werden. Wichtig dabei ist, dass das JOBLIB-Statement nach dem JOB-Statement und vor dem ersten EXEC-Statement steht.

Das DD-Statement veranlasst das Verbinden der Systembibliothek mit einer privaten Bibliothek. Die Bilbliotheken werden dabei verkettet. Zuerst wird die private, dann werden die im System definierten Bibliotheken durchsucht.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T
//*
//STEP1    EXEC PGM=IEFBR14
//STEPLIB  DD DSN=SYS1.IBM.LOADLIB,
//            DISP=SHR
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            DSORG=PS,
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 36: STEPLIB-Angabe

Diese Verkettung kann auch auf einzelne Steps beschränkt werden, indem für den betroffenen Step in 'STEPLIB DD-Statement' geschrieben werden.

```
…
//*
//STEP1    EXEC PGM=IEFBR14
//…
//*
//STEP2    EXEC PGM=PRIVPROG
//STEPLIB  DD DSN=USERHLQ1.USERHLQ2.USERHLQ3,
//            DISP=SHR
//*
//STEP3    EXEC PGM=IEBCOPY
//…
…
```

Abb. 37: STEPLIB im zweiten von drei Steps

Die Programme IEFBR14 und IEBCOPY in Abbildung 37 werden nur in den Systembibliotheken gesucht, da weder ein JOBLIB- noch ein STEPLIB-Statement angegeben wurde. IDCAMS wird zunächst in der privaten Bibliothek

gesucht, da für diesen Step ein STEPLIB-Statement kodiert wurde. Bei 'Nichtgefunden' wird anschließend auch die Systembibliothek durchsucht.

3.9 Das DD-Statement SYSIN

Wenn die Job Control Daten enthält, erwartet das JES direkt vor diesen Daten ein Statement 'DD *' oder 'DD DATA'. Sollte dieses Statement nicht vorhanden sein, wird vom JES automatisch ein entsprechendes DD-Statement mit dem DD-Namen SYSIN generiert.

Wenn also in der Ausgabe eines Jobs der Text

- SYSIN – GENERATED STATEMENT

auftaucht und nicht klar ist, wieso SYSIN generiert wurde, kann das verschiedene Ursachen haben:

- Die In-Stream-Daten in der Job Control wurden an die falsche Stelle geparkt und ein DD-Statement mit 'DD *' oder 'DD DATA' steht an einer anderen Stelle einsam und verlassen in der Gegend herum.
- Das DD-Statement wurde vergessen zu kodieren.
- Es wurde versäumt, vor einem anderen JCL-Statement die Schrägstriche '//' zu kodieren, was zur Folge hat, dass das JES diese Zeile als Daten, zu denen das DD-Statement fehlt, interpretiert.

3.10 Concatenated Datasets

In der Verarbeitung gibt es immer wieder Situationen, in denen mehrere Datenbestände nacheinander durch

- ein (☝) Programm

und unter

- einem (☝) Namen

verarbeitet werden sollen. Zur Lösung dieses Problems gibt es zwei Ansätze:

- Die brutale Methode: Die Datenbestände werden hintereinander kopiert und als Gesamtdatenbestand verarbeitet.
- Die elegante Methode: Es werden Concatenated Datasets verwendet.

Wie bereits erwähnt wird durch das Betriebssystem immer ein Block vom externen Datenbestand in den Puffer übertragen. Dies geschieht solange, bis der letzte Block des Datenbestandes sich im Puffer befindet. Fordert das Betriebssystem danach noch einen weiteren Block an, wird vom Datenträger ein 'END OF DATA' (EOD) zum Puffer übertragen. Das Betriebssystem weiß damit, dass dieser Datenbestand zu Ende ist.

Das Betriebssystem schaut nun nach, ob noch ein weiterer Datenbestand mit diesem DD-Namen verknüpft ist. Ist dies der Fall, so schaltet das Betriebssystem zum neuen Datenbestand um und überträgt weitere Blöcke von diesem Datenträger in den Puffer.

Der Prozess läuft solange, bis auch dieser Datenbestand zu Ende ist. Anschließend wiederholt sich dieser Vorgang für den nächsten Datenbestand usw., bis der letzte Datenbestand abgearbeitet ist.

Ist das letzte EOD durch das Betriebssystem erkannt und steht kein weiterer Datenbestand zur Verarbeitung zur Verfügung, dann übergibt das Betriebssystem ein 'END OF FILE' an das Programm. Das folgende Beispiel zeigt die Syntax zur Anwendung von Concatenated Datasets:

```
…
//INFILE   DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=SHR
//         DD DSN=HLQ4.HLQ5.HLQ6,
//            DISP=SHR
//         DD DSN=HLQ7.HLQ8.HLQ9,
//            DISP=SHR
…
```

Abb. 38: Concatenated Datasets

Wie in Abbildung 38 dargestellt dürfen die DD-Statements der Datenbestände, die an den ersten Datenbestand gekettet werden sollen, kein Name-Field haben. Haben die Datenbestände unterschiedliche Blocklängen, muss der Datenbestand mit der größten Blockung als erster aufgeführt werden.

Zu beachten ist an dieser Stelle auch, dass es vom jeweiligen Programm abhängt, ob dieses mit einer derartigen Kodierung arbeiten kann. Es ist nicht garantiert, dass jedes Programm mit Concatenated Datasets arbeiten kann.

Das Betriebssystem verarbeitet nur den DCB des ersten DD-Statements. Aus dem DCB leitet es die Größen der internen Puffer ab, in die die Datenbestände eingelesen werden. Liegt nun der Datenbestand mit der größten Blocklänge nicht auf dem ersten DD-Statement, dann reicht der Lesepuffer für den Zugriff auf diesen Datenbestand nicht aus.

Partitioned Datasets (PO-Dateien) können ebenfalls verkettet werden. Das bedeutet, dass mehrere PDS einem DD-Statement zugeordnet werden. Das Suchen eines Members läuft dann gerade so, als bildeten die verketteten PDS ein einziges Dataset.

Der Suchvorgang im concatenated PDS betrachtet ein gemeinsames Verzeichnis, das von oben durchsucht wird. Sobald ein Member gefunden wird, bricht der Suchvorgang ab. Sollte ein Member in mehreren PDS unter gleichem Namen existieren, so wird das Member aus dem PDS genommen, das in der Suchfolge obenan steht.

3.11 Das NULL-Statement

Dieses Statement wird vor allem zu Testzwecken verwendet. Soll ein Job mit zehn Steps nur bis zum fünften Step laufen, können die restlichen Steps entweder herausgelöscht oder aber nach dem fünften Step ein NULL-Statement gesetzt werden.

```
…
//*
//STEP1    EXEC PGM=IEFBR14
//…
//*
//STEP2    EXEC PGM=IEBGENER
//…
//*
//
//STEP3    EXEC PGM=IEBCOPY
//…
…
```

Abb. 39: NULL-Statement

Der STEP3 in Abbildung 39 und alle nachfolgenden Steps dieses Jobs werden nicht mehr ausgeführt.

3.12 Der JCL-Error

Es gibt zwei unterschiedliche Arten, einen JCL-Error (JCLERR) zur erzeugen. Zum einen während der syntaktischen durch das JES[6], zum anderen während eines Joblaufes. Wie ist das möglich?

Wie in Kapitel 2.6 beschrieben prüft das JES die JCL auf syntaktische Richtigkeit, d.h.

→ stehen die Parameterwerte in den richtigen Feldern
→ wurden keine Kommata vergessen
→ sind alle Angabe (bis auf Vorlaufkarten) in Großbuchstaben geschrieben
→ usw.

Weist ein Job derartige Fehler auf, Dokumentiert das System über den Job-Log[7] und das Sys-Message-Log[8]. Die folgende Abbildung verdeutlicht dies:

[6] vgl. Kapitel 2.6
[7] vgl. Kapitel 10.2
[8] vgl. Kapitel 10.3

```
//JOB00001 JOB (123456),'SCHEIBE M',CLASS=A,
//             MSGCLASS=T,
//             COND=(0,NE),
//             NOTIFY=&SYSUID.,
//*
*************************** ILLEGAL CONTINUATION ********************
$HASP106 JOB DELETED BY JES2 OR CANCELLED BY OPERATOR BEFORE EXECUTION
```

Abb. 40: Fehlermeldung bei Syntaxfehler

In dem Fall, welcher in Abbildung 40 aufgetreten ist geht nicht genau hervor, aus welchem Grund der Job JOB0001 eliminiert wurde. Es wird nur die Aussage getroffen, dass der Job vor der Ausführung entweder durch das JES2 oder einem Operator aus dem System entfernt wurde. Im Ergebnis heißt das: Fehlersuchen. In diesem Fall ist die Fehlerursache darin begründet, dass in der Jobkarte an der Angabe „NOTIFY=&SYSUID.“ ein Kommata angefügt wurde. Diese Angabe ist syntaktisch falsch. Dadurch wurde der Jobabbruch verursacht.

Was innerhalb dieser JES2-Kontrolle nicht abgedeckt wird ist die Prüfung auf im Job angesprochene DSNNAMES[9]. Wird ein Job auf der Maschine zur Ausführung gebracht und kommt er während der Ausführung in ein EXEC-Statement, also einen Step, in dem eine Datei angesprochen wird, die nicht vorhanden ist, bricht der Job auch mit JCLERR ab. Dieser Fehler hat jedoch nichts mit der Plausibilitätskontrolle durch das JES2 zu tun, sondern mit einem fehlenden Katalogeintrag[10].

Die folgende Abbildung zeigt das Reporting eines solchen Fehlers im Job-Log und im Sys-Message-Log:

```
…
-JOBNAME  STEPNAME PROCSTEP    RC   EXCP
-JOB00001 STEP1                00     20
-JOB00001 STEP2              FLUSH     0
IEF453I JOB00001 - JOB FAILED - JCL ERROR
…
```

Abb. 41: Job-Log mit RC=FLUSH

```
JOB00001 STEP2 SYSUT1 - DATA SET NOT FOUND
JOB00001 STEP2 - STEP WAS NOT EXECUTED.
```

Abb. 42: Sys-Message-Log mit Begründung für FLUSH

Abbildung 41 zeigt an, dass STEP1 ausgeführt und mit RC 00 beendet wurde, STEP2 jedoch nicht ausgeführt wurde. Abbildung 42 dokumentiert die Begründung für den FLUSH.

[9] vgl. Kapitel 3.2
[10] vgl. Kapitel 1.2.1

3.13 JCL-Variablen

Eine Variable ist ein Platzhalter innerhalb eines (Programm-)Ablaufes für einen Wert, der zu Beginn noch nicht feststeht. Die Variable bekommt erst während der aktiven Verarbeitung einen eindeutigen Wert zugewiesen. In vielen Programmiersprachen ist es erforderlich, die Variable vor der eigentlichen Wertezuweisung mit sog. Metainformationen zu definieren.

Wie in Kapitel 1.3 bereits angesprochen können auch in der JCL Variablen verarbeitet werden. Anders als in anderen Programmiersprachen wie z.B. VBA ist hier die Angabe von Metainformationen nicht erforderlich. Eine JCL-Variable wird über den SET-Parameter im Operationfield definiert.

In der JCL können zwei verschiedene Variablentypen definiert werden:

- der **erste Variablentyp** ('&var1.') beginnt mit einem Ampersand ('&') und endet mit einem Punkt ('.'). Die Gültigkeit dieses Variablentyps beginnt mit dem Punkt der Definition und endet mit dem Job.

  Der Punkt, an dem die Variable definiert wird muss nicht zwingend zwischen Jobkarte und dem ersten EXEC-Statement liegen. Die Variable kann auch zwischen zwei EXEC-Statements definiert werden. Es ist auch möglich, die Variable zu Beginn eines Jobs zu definieren und im weiteren Verlauf des Jobs zwischen zwei EXEC-Statements einen neuen Wert zuzuweisen.

  Dieser Variablentyp ist in der Regel ein Bestandteil des Dateinamens, wenn ein Qualifier dynamisch vom Betriebssystem ermittelt werden muss.

- der **zweite Variablentyp** ('&&var1') beginnt mit zwei Ampersands ('&&'). Die Gültigkeit dieses Variablentyps beginnt und endet mit dem Step, in welchem er definiert wurde.

  Dieser Variablentyp wird in der Regel verwendet, wenn man sich die Angabe eines vollqualifizierten Dateinamens ersparen will.

Der erste Variablentyp

In der Praxis sieht eine einfache Variablenverarbeitung wie folgt aus:

```
…
//         SET VAR1='WERT1'
//*
//STEP1    EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.&VAR1.,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 43: Einfache Variablenverarbeitung

Abbildung 43 zeigt den Einsatz der Variable 'VAR1'. Diese Variable wird mit dem Wert 'WERT1' belegt und in den dritten Qualifier des Dateinamens in STEP1 eingesetzt. Der Dateiname sieht nach der Variablenauflösung so aus:

- HLQ1.HLQ2.**WERT1**

Man kann die in Abbildung 43 dargestellte JCL auch wie folgt aufbauen, ohne dass es während der Plausibilitätskontrolle durch das JES2 zu Problemen führen würde:

```
…
//          SET VAR1='WERT1'
//*
//STEP1     EXEC PGM=IEFBR14
//DD01      DD DSN=HLQ1.&VAR1.FIL.HLQ3,
//             DISP=(NEW,CATLG,DELETE),
//             RECFM=FB,
//             LRECL=80,
//             SPACE=(TRK,(1,10000),RLSE)
```

Abb. 44: Einfache Variablenverarbeitung (Alternative)

Wie man in den Abbildungen 43 und 44 gut erkennen kann, kann der Punkt am Ende einer Variablen weggelassen werden. Dies ist jedoch nur dann der Fall, wenn die Variable am Ende eines Dateinamens steht und keine weiteren Qualifier folgen.

Möchte man den Wert der Variablen + einen weiteren, variablen oder konstanten Wert konsolidieren, ist der Punkt nach der Variablen erforderlich. Die folgende Abbildung verdeutlicht dies:

```
…
//          SET VAR1='WERT1'
//*
//STEP1     EXEC PGM=IEFBR14
//DD01      DD DSN=HLQ1.&VAR1..FIL,
//             DISP=(NEW,CATLG,DELETE),
//             RECFM=FB,
//             LRECL=80,
//             SPACE=(TRK,(1,10000),RLSE)
```

Abb. 45: Zusammenspielen einer Variablen und einer Konstanten

In der Auflösung der in Abbildung 45 dargestellten JCL sieht der Dateiname so aus:

- HLQ1.**WERT1FIL**.HLQ3

Dies funktioniert jedoch nur dann, wenn der zugeordnete Wert der Variablen und der konstante Wert zusammen nicht mehr als acht Zeichen haben (siehe Kapitel 1.2.1), da anderweitig folgende Fehlermeldung ausgegeben wird:

- EXCESSIVE PARAMETER LENGTH IN THE DSNAME FIELD

Nun gibt es noch die Variante, den Wert einer Variablen als eigenen HLQ laufen zu lassen. In diesem Fall muss die Variable mit zwei Punkten beendet

werden. Der erste Punkt dient dazu, die Variable, der zweite um den Qualifier zu beenden. Die folgende Abbildung veranschaulicht dies:

```
…
//          SET VAR1='WERT1'
//*
//STEP1     EXEC PGM=IEFBR14
//DD01      DD DSN=HLQ1.&VAR1..FIL,
//             DISP=(NEW,CATLG,DELETE),
//             RECFM=FB,
//             LRECL=80,
//             SPACE=(TRK,(1,10000),RLSE)
```

Abb. 46: Trennen einer Variablen und einer Konstanten

In der Auflösung der in Abbildung 46 dargestellten JCL sieht der Dateiname so aus:

- HLQ1.**WERT1**.FIL

Der zweite Variablentyp

Umgangssprachlich wird dieser Variablentyp auch als 'temporäre Variable' oder 'temp Variable' bezeichnet. Der Einsatz einer solchen Variablen sieht in der Praxis so aus:

```
…
//STEP1     EXEC PGM=IEFBR14
//DD01      DD DSN=&&VARIABLE1,
//             DISP=(NEW,CATLG,DELETE),
//             RECFM=FB,
//             LRECL=80,
//             SPACE=(TRK,(1,10000),RLSE)
```

Abb. 47: Verwendung einer Temp-Variablen

Eine Temp-Variable ist im Grunde genommen genauso zu behandeln wie ein vollqualifizierter Dateiname. Für diese Variabletypen wird dynamisch durch das Betriebssystem ein vollqualifizierter Dateiname vergeben und am Ende des Joblaufes wieder gelöscht.

4 Job Processing

Ein Job wird in fünf Schritten durch das System geschleust:

- 1. Input
- 2. Conversion
- 3. Job-Execution
- 4. Output
- 5. Purge

Durch einen Übergabebefehl, im Dialog z.B. Sub, wird dem JES ein Auftrag übergeben. Die Job Control wird durch ein spezielles Programm, dem READER, verarbeitet. Es gibt verschiedene READER, abhängig davon, von wo aus die Job Control gestartet wird.

Das Symbol für Job Control ist eine Lockkarte. Das hängt damit zusammen, dass Job-Control-Befehle 80-stellig geschrieben werden und sehr stark an das ursprüngliche Lochkartenformat angelehnt sind.

Die Job Control kann auch Daten enthalten. Diese werden von den Steueranweisungen getrennt und als eigener Datenbestand in die SPOOL gestellt. Die SPOOL ist die Datei, in die alles eingestellt wird, was für einen Auftrag vom JES erzeugt wird. SPOOL ist eine Abkürzung und bedeutet:

- **S**imultaneous **P**eripheral **O**perations **O**n **L**ine.

4.1 Input

Was alles zu einem Auftrag gehört, wird mit Hilfe des Job Queue Element festgehalten. Die SPOOL ist aber nicht nur eine Datei, sie stellt die technische Umsetzung einer Queue (Warteschlange) dar. Die für den Auftrag erzeugten Dateien bleiben solange in der Input-Queue, bis sie durch Beenden des Auftrages, ob erfolgreich oder nicht, in die Output-Queue gestellt werden.

4.2 Conversion

Im nächsten Schritt des JES wird die Job Control für die Verarbeitung durch das MVS aufbereitet. Man spricht von einer Conversion oder auch von einer Interpretation. Dieser Vorgang beinhaltet folgende Prozesse:

- Die Job Control wird auf sogenannte Prozeduraufrufe überprüft. Diese separat abgelegten Arbeitsanweisungen werden gegebenenfalls zu diesem Zeitpunkt eingebunden

- Die Job Control wird auf ihre formale Richtigkeit überprüft, d.h. der Job Stream wird auf formale Fehler geprüft.

- Ist der Job Stream fehlerfrei wird er übersetzt und das Übersetzungsergebnis als Internal Text in die SPOOL abgestellt

- Wird während der Konvertierung ein Fehler festgestellt, dann wird der folgende Schritt übersprungen:

  Der Job landet in der Output Queue. Dort bleibt er, bis er gedruckt oder vom Anwender entfernt wird.

4.3 Job Execution

Die eigentliche Bearbeitung des Jobs wird jetzt vom JES an das MVS weitergegeben. Die Ausführung wird nicht vom JES angestoßen, sondern von einem Initiator[11].

4.4 Output

In der Output Phase werden alle während der Ausführung erzeugten und auf SPOOL zwischengespeicherten Ausgaben an die Ausgabegeräte, die in der JCL angegeben sind, kopiert.

Der Job hat nun sein Ziel erreicht. Die Arbeit des JES ist damit aber noch nicht ganz erledigt.

4.5 Purge

Wenn alle Ausgaben des Auftrags ihre Zielgeräte erreicht haben, wird im Spoolbereich aufgeräumt, d.h. in der Purge Phase werden alle Dateien, die für diesen Auftrag auf der Spool angelegt worden sind, gelöscht.

Diese fünf Schritte sind Bestandteil des JES.

Es werden also bei der Durchführung eines Jobs zwei verschiedene Systeme angesprochen:

Das JES und das Betriebssystem.

Das JES setzt die Arbeitsaufträge in eine für das Betriebssystem verständliche Form um. Es ist praktisch der Dolmetscher zwischen dem Anwender und dem Betriebssystem.

[11] Siehe Unterkapitel 2.4 (Job-Klasse)

5 Notation

Um die Struktur von JCL Statements deutlich zu machen wird eine genormte Schreibweise verwendet. Hier spricht man von der Notation, in der Großbuchstaben und Kleinbuchstaben sowie Sonderzeichen bestimmte Funktionen haben.

5.1 Großbuchstaben

Um die Sache zu vereinfachen sehen wir uns noch einmal unseren ersten Job an:

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//FILECOPY EXEC PGM=IEBGENER
//SYSUT1   DD DSN=HLQ0.HLQ1.HLQ2,
//            DISP=SHR
//SYSUT2   DD DSN=HLQ10.HLQ11.HLQ12,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=*.SYSUT1,
//            SPACE=(TRK,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb. 48: Die Notation des ersten Jobs

Wie in der Abbildung 48 zu erkennen ist, sind alle Buchstaben des Jobs groß geschrieben, denn grundsätzlich gilt folgende Regel:

- Job Control wird immer in Großbuchstaben geschrieben.

5.2 Kleinbuchstaben

Die Kleinschreibung eines Begriffes bedeutet, dass hier eine Information eingesetzt wird. Diese Information kann weiteren Regeln unterworfen sein:

- z.B. ein Wert aus einem Wertevorrat

oder

- Konventionen über Namensvergaben.

In den Fällen, in denen Kleinbuchstaben geschrieben werden, kann man davon ausgehen, dass es sich um einen Datenstrom oder um eine Kommentarzeile handelt. Die folgenden Beispiele verdeutlichen dies:

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//FILECOPY EXEC PGM=IEBGENER
//SYSUT1   DD *
Dies sind Daten für die
Ausgabedatei. Anhand dieses Beispiels
soll verdeutlicht werden, unter welchen
Voraussetzungen in einer JCL Kleinbuch-
staben zulässig sind.
//SYSUT2   DD DSN=HLQ10.HLQ11.HLQ12,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=*.SYSUT1,
//            SPACE=(TRK,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb. 49: Job mit Datenstromkarte

Wie wir bereits in Kapitel 2.4 festgestellt haben, ist im Programm IEBGENER das DD-Statement SYSUT1 fest als Eingabedatenstrom definiert. Das Bedeutet, dass alle Informationen, die über SYSUT1 kommen, als Eingabedaten gelesen werden.

Nun ist es völlig gleichgültig, ob diese Daten per Datei zu Verfügung gestellt oder wie in Abbildung 7 über eine sogenannte Vorlaufkarte vom Anwender eingegeben werden.

Sobald der Job submittet (gestartet) wird, werden die Eingabedaten unter SYSUT1 in die Ausgabedatei unter SYSUT2 geschrieben und auf einem Speichermedium abgelegt.

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//*Dieser Step kopiert eine Datei
//FILECOPY EXEC PGM=IEBGENER
//SYSUT1   DD DSN=HLQ0.HLQ1.HLQ2,
//            DISP=SHR
//SYSUT2   DD DSN=HLQ10.HLQ11.HLQ12,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=*.SYSUT1,
//            SPACE=(TRK,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb. 50: Job mit Stepbeschreibung

Wie aus Abbildung 50 ersichtlich wird, können auch Kommentarzeilen in Kleinbuchstaben geschrieben werden. In beiden Fällen handelt es sich um Informationen und nicht um JCL-Anweisungen.

5.3 Geschweifte Klammern

Die geschweiften Klammern umschließen einen Ausdruck, der angegeben werden muss. Der Ausdruck selbst besteht aus einer Menge von Ausdrücken, aus der genau ein Element auszuwählen ist.

5.4 Senkrechter Balken

Der senkrechte Balken (|) ist ein weiteres Sonderzeichen. Es besagt, dass aus einem Wertevorrat genau ein Element auszuwählen ist. Das Zeichen steht dabei für ein Exclusives Oder (XOR).

5.5 Eckige Klammern

Die eckigen Klammern umschließen einen Ausdruck, der angegeben werden kann.

```
//ddname DD [positional-parameter]
//          [keyword-parameter]
```

Abb. 51: JCL mit eckiger Klammer

Der Begriff „keyword-parameter“ kann, wie in Abbildung 51 dargestellt, kodiert werden, wenn es der Programmierer für nötig erachtet. Eine reale Job Control würde wie folgt aussehen:

```
//DD01      DD DSN=HLQ1.HLQ2.HLQ3,
//             DISP=SHR
```

Abb. 52: Eine reale Job Control

5.6 Unterstreichung

In einem Wertevorrat wird der voreingestellte Wert, der sogenannte Default, unterstrichen. Wenn Sie diesen Wert benutzen wollen, brauchen Sie diesen Parameter nicht zu kodieren.

- ADDRSPC=(VIRT|REAL)

ADDRSPC=VIRT brauchen Sie nicht zu kodieren, weil VIRT der Voreinstellung von ADDRSPC entspricht. In manchen Fällen kann es trotzdem vorteilhaft sein, die Angabe mit der Voreinstellung zu schreiben. Es gibt Parameter, die so selten gebraucht werden, dass niemand die Voreinstellung im Kopf hat.

6 Formalitäten

6.1 Erläuterungen

Die Job Control ist eine formatgebundene Sprache, d.h. die Befehle dieser Sprache sind in einzelne Felder untergliedert. Die Reihenfolge dieser Felder innerhalb eines Befehls ist genau festgelegt.

Job Control gab es schon zu Zeiten, als noch mit Lochkarten gearbeitet wurde. Daher wird Job Control in Abbildungen und Graphiken häufig als Lochkarte dargestellt. Aus dieser Vergangenheit ist es zu erklären, dass die JCL eine formatgebundene Sprache ist.

6.2 Die allgemeine Syntax

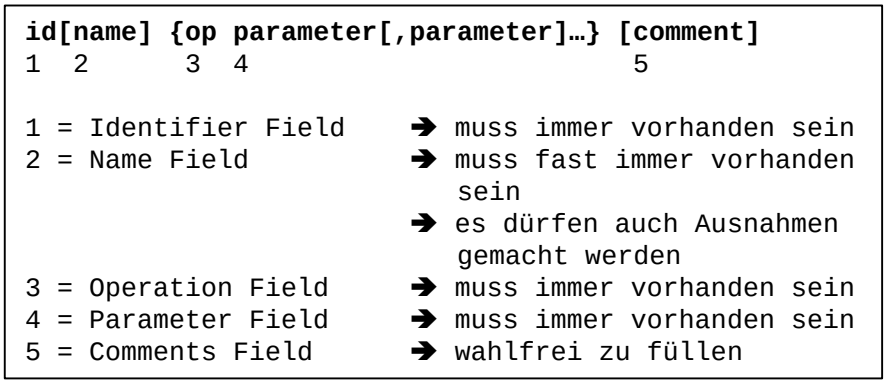

Abb. 53: Die allgemeine Syntax

Wie aus Abbildung 53 ersichtlich wird, kann die allgemeine Syntax eines JCL-Statements aus bis zu fünf Feldern bestehen. Zwischen

- Name Field und Operation Field,
- Operation Field und Parameter Field,
- Parameter Field und Comment Field

muss mindestens ein Leerzeichen eingefügt werden, es können aber auch mehrere Leerzeichen sein. Wenn man sich die folgende Zeile

id[name] {op parameter**[,parameter]…**} [comment]

genauer ansieht stellt man anhand der drei Punkte fest, dass [,parameter] beliebig oft wiederholt werden darf, je nachdem, wie Parameterangaben für dieses Statement vorhanden und auch sinnvoll sind.

Was aber tun, wenn die 71. Spalte erreicht ist? Auch für dieses Problem gibt es eine Lösung. Wenn mehr Parameter benötigt werden, können die Statements in weiteren Zeilen fortgesetzt werden. Bei einer Fortsetzung muss folgendes beachtet werden:

- eine Zeile wird durch ein Kommata beendet
- die nächste Zeile wird, wie gewohnt, mit zwei Schrägstrichen begonnen
- die Kodierung muss **frühestens ab Spalte vier, spätestens ab Spalte 16** fortgesetzt werden

Die Fortsetzungszeile darf nicht mit einem Kommata versehen werden, da das Kommata am Ende der vorhergehenden Zeile geschrieben wird. Diese Angaben sind obligatorisch. Schreibt der JCL-Programmierer trotzdem ein Komma am Beginn der Fortsetzungszeile und lässt beispielsweise das Kommata am Ende der vorhergehenden Zeile weg, wird des Job während der Plausibilitätskontrolle durch das JES storniert und mit dem Hinweis „JCL ERROR“ in die Output-Facility gestellt. In der System-Message erscheint folgende Fehlermeldung:

- POSITIONAL PARAMETERS MUST BE SPECIFIED BEFORE KEYWORD PARAMETERS

Schreibt der JCL-Programmierer allerdings nur in der Fortsetzungszeile ein Kommata und lässt es in der vorhergehenden Zeile weg, wird der Job auch während der Plausibilitätskontrolle durch das JES storniert und mit JCL ERROR in die Output-Facility gestellt. In der System-Message erscheint dann allerdings folgende Fehlermeldung:

- UNIDENTIFIED OPERATION FIELD

Das ist darin begründet, dass eine Zeile durch ein weggelassenes Kommata beendet wird. Es folgen keine weiteren Angaben, welche als Fortsetzung zu deuten sind. Das JES kann also eine Zeile, die nur durch „//“ begonnen wird und auf eine Zeile ohne Kommata folgt, nicht zuordnen. Wird also eine Zeile durch ein weggelassenes Kommata beendet, muss die Folgezeile durch ein DD-Statement eingeleitet werden.

Wie bereits erwähnt ist Job Control eine formatgebundene Sprache. Die Formatbindung bezieht sich jedoch nur auf die Einhaltung der Felder und auf die Angaben von JCL-Kommandos in Großbuchstaben. Bei der Angabe von Metainformationen während der Katalogisierung eines Datasets beispielsweise existieren keine Vorgaben. Das folgende Beispiel zeigt, wie eine solche JCL aussehen könnte:

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//STEP1 EXEC PGM=IEFBR14
//DD01 DD DSN=HLQ10.HLQ11.HLQ12,
//  DISP=(NEW,CATLG,DELETE),RECFM=FB,RECL=80,
//     SPACE=(TRK,(1,10000),RLSE)
```

Abb. 54: Katalogisierung eines Datasets (unstrukturiert)

Die Abbildung 54 ist ein Negativbeispiel für eine Job Control, wie man es besser nicht machen sollte. Die dargestellte Job Control ist unstrukturiert und

unübersichtlich. Die folgende Abbildung zeigt ein Beispiel, wie eine JCL aufgebaut werden sollte:

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14 EXEC PGM=IEFBR14
//DD01 DD DSN=HLQ10.HLQ11.HLQ12,
//        DISP=(NEW,CATLG,DELETE),
//        RECFM=FB,
//        LRECL=80,
//        SPACE=(TRK,(1,10000),RLSE)
```

Abb. 55: Katalogisierung eines Datasets (strukturiert)

Anhand der beiden in Abbildung 54 und 55 dargestellten Beispiele wird deutlich, dass es zwar syntaktisch korrekt wäre, eine Zeile bis zur Spalte 71 durch zu schreiben, dies die Fehlersuche jedoch erschwert. Das Beispiel in Abbildung 55 ist zwar nicht unbedingt platzsparend, dafür aber strukturiert und übersichtlich.

6.3 Das Identifier Field

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 56: Das Identifier Field

Das Identifier Field, in Abbildung 56 Fett gedruckt, steht in den Spalten eins und zwei und unterscheidet die JCL Statements von Daten. Daten können zwischen den einzelnen Statements stehen, wie das folgende Beispiel zeigt:

```
JCL Statement → //JOB00001 JOB (123456),SCHEIBE,CLASS=A,
“   “         → //             MSGCLASS=T,NOTIFY=&SYSUID
“   “         → //*
“   “         → //IDCAMS   EXEC PGM=IDCAMS
“   “         → //SYSIN    DD *
Daten         →   DELETE HLQ1.HLQ2.HLQ3
JCL Statement → //SYSPRINT DD SYSOUT=*
“   “         → //*
“   “         → //IEFBR14  EXEC PGM=IEFBR14
“   “         → //DD01     DD DSN=HLQ1.HLQ2.HLQ3,
“   “         → //            DISP=(NEW,CATLG,DELETE),
“   “         → //            RECFM=FB,
“   “         → //            LRECL=80,
“   “         → //            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 57: Unterschied JCL Statement / Daten

Fazit:

- ein “//“ im Identifier Field kennzeichnet ein JCL Statement
- ein “/*“ im Identifier Field kennzeichnet das Ende von Daten oder ein JES2 Statement
- alle anderen Zeichen im Identifier Field kennzeichnen die Zeile als Datenzeile

6.4 Das Name Field

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 58: Das Name Field

Wie in Abbildung 58 zu erkennen ist, beginnt das Name Field direkt hinter dem Identifier Field auf Spalte drei und gibt dem Statement einen Namen. Bei der Namensvergabe sind folgende Regeln zu beachten:

- der Name darf höchstens acht Stellen lang sein
- er muss mit einem Buchstaben (A, B, C, …) oder einem National Character ($, #, @) beginnen
- die folgenden Zeichen können aus Buchstaben, National Character oder Ziffern bestehen

Es gibt noch eine Besonderheit im Name Field:

Wie in anderen Programmiersprachen kann auch in der JCL eine Kommentarzeile eingefügt werden. Kommentarzeilen fügt der Programmierer ein, um den Programmablauf zu dokumentieren. Die eine Möglichkeit, Kommentarzeile zu erstellen, besteht darin, Identifier Field mit Slash (//) und das Name Field mit Asterik (*) zu füllen, wie aus Abbildung 17 hervorgeht. Es handelt sich hierbei um reine Kommentarzeilen. Das Comment Field wird in Unterkapitel 5.7 „Das Comment Field“ ausführlich erläutert.

Vorsicht: JES3 Statements beginnen ebenfalls mit der Zeichenkette „//*“.

Das Comment Field wird in Abschnit 5.6 „Das Comment Field“ ausführlich erläutert.

6.5 Das Operation Field

Das Operation Field gibt den Typ des JCL Statements an. Vor dem Operation Field muss immer eine Leerstelle sein, auch dann, wenn das Name Field nicht besetzt ist.

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 59: Das Operation Field

6.6 Das Parameter Field

Der Operand besteht aus zwei Arten von Parametern:

- Positions-Parameter
- Keyword-Parameter (Schlüsselwort-Parameter)

Positionsparamter sind durch ihre Position in Bezug auf andere Parameter gekennzeichnet[12]. Beim Job Statement gibt es zwei Positionsparameter:

- accounting
- programmers name

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 60: Das Parameter Field unter Hervorhebung der Positions-Paramter

Die Angabe des „accounting“ dient dazu, den Joblauf dem entsprechenden Kunden in Rechnung stellen zu können.

Die Angabe des „programmers name“ wird benötigt, um den Namen des Programmierers in den Vorspann der Jobausgabe zu drucken. Dieser Parameter ist im Prinzip als Begriff zu sehen und darf maximal 20 Stellen lang sein. Wird dieser Begriff durchgängig geschrieben, also z.B. SCHEIBE, dann kann er ohne Eingrenzungszeichen geschrieben werden. Andernfalls ist der Begriff durch Hochkommata oder Klammer einzugrenzen, da die weiteren

[12] vgl. G.D.Brown: JCL Job Control Language im Betriebssystem OS/390 MVS, 3. Aufl., S, 65

Buchstaben sonst fälschlicher Weise vom JES als Kommentar interpretiert werden.

```
//JOB00001 JOB (123456),'M SCHEIBE',CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 61: Programmers Name in Hochkommata

Will man bei einer Folge von Positionsparametern einzelne auslassen, z.B. weil die Standardeinstellungen ausreichen, dann muss für diesen ausgelassenen Parameter ein Platzhalter verwendet werden. Dies geschieht durch die Angabe eines Kommatas. Abbildung 21 macht dies deutlich.

```
//JOB00001 JOB ,SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 62: Das Parameter Field unter Hervorhebung der Positions-Parameter und ohne Accounting

Keyword-Parameter dürfen in beliebiger Reihenfolge stehen. Erkennbar sind Schlüsselwortparameter an dem Gleichheitszeichen, das dem Parameter direkt folgt. Die Keyword-Parameter werden, im Gegensatz zum Positionsparameter, in beliebiger Reihenfolge eingetragen, aber immer erst nach den positionellen Parametern.

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 63: Das Parameter Field unter Hervorhebung der Keyword-Parameter

CLASS=A ist ein solcher keyword-parameter. Erkennbar sind Schlüsselwortparameter an dem Gleichheitszeichen, das dem Parameter direkt folgt. Die keywort-parameter werden, im Gegensatz zum Positionsparameter, in beliebiger Reihenfolge eingetragen, aber immer erst nach den positionellen Parametern.

Subparameter

Bestimmte Positionsparameter, sowie die variable Information eines Schlüsselwortparameters können aus mehreren Begriffen, den sogenannten Subparametern bestehen. Die Subparameter einer Liste können wiederum Positions- und Schlüsselwortparameter sein.

- **Für Subparameter gelten die gleichen Regeln wie für Parameter**

Zur Unterscheidung der einzelnen Parameter werden deshalb Sonderzeichen benutzt. Diese Sonderzeichen tauchen im JCL-Code selbst auf, sie sind also nicht bloß Notation. Sonderzeichen bestehen aus

- () runden Klammern
- ,' Apostrophe

Die runden Klammern umrunden eine Liste von Subparametern, deren Reihenfolge genau festgelegt ist. Das bedeutet, dass die Begriffe nur über ihre Position in der Liste erkannt werden. Werden mehrere Subparameter angegeben, so sind sie in runde Klammern zu setzen und durch Kommata voneinander zu trennen.

- **Anstelle der Klammern können auch Apostrophe gesetzt werden. Die Klammern dürfen fehlen, wenn nur ein Subparameter vorhanden ist.**

```
//JOB00001 JOB (1234,2345,3456,,,,,,,7890),
//             'SCHEIBE',
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 64: Das Parameter Field mit mehreren Account-Nummern

In Abbildung 64 wurde als „accounting" eine Liste von Subparametern angegeben. Daher müssen diese Angaben eingeklammert werden. Die Kommata innerhalb der Klammer stehen als Platzhalter für nichtkodierte Subparameter.

Die Apostrophe, auch Hochkommata genannt, erfüllen in der Job Control zwei Aufgaben:

1. Sie können, wie oben erwähnt, eine Liste von Subparametern umschliessen.
2. Sie müssen Wertangaben umschliessen, wenn diese Sonderzeichen enthalten.

So muss also, wie beispielsweise in Abbildung 21 in Unterkapitel 5.6 „Das Parameter Field" dargestellt, der „programmers name" in Apostroph

geschrieben werden, wenn die Angabe einen Blank enthält. Enthält der Name eines Programmierers selbst ein Hochkommata, z.B. O'Connors, so muss dieses doppelt kodiert werden.

```
//JOB00001 JOB (123456),'O''Conners',CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 65: Hochkommata innerhalb eines Programmierernamens

6.7 Das Comment Field

Auf das Parameter Field kann nach mindestens einem Leerzeichen das Comment Field folgen, wie das nachfolgende Beispiel zeigt. Es enthält Informationen, die der Programmierer für sinnvoll erachtet. Sie werden nicht verarbeitet und dienen nur als Kommentar.

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A, Kommentar
//             MSGCLASS=T,NOTIFY=&SYSUID Kommentar
//*
//*************
//* Kommentar *
//*************
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 66: Das Comment Field

Die Zeilen drei bis sechs in Abbildung 66 zeigen, wie Kommentarzeilen noch eingefügt werden können. Es handelt sich dabei um das Comment Statement mit der Form:

- //* comments

Reicht die Zeile für den Text nicht aus, werden mehrere Statements geschrieben, da das Comment Statement kein Fortsetzungszeichen in Spalte 72 kennt. Bei der Auswahl des Textes ist der Programmierer allerdings völlig ungebunden, da, im Gegensatz zu JCL Statement, Kleinbuchstaben erlaubt sind.

Die Schreibweise des Kommentars als Anhängsel hinter den Parametern ist ein Relikt aus der Zeit der Lochkarte. Heute werden allerdings Comment Statements eingesetzt.

Hervorzuheben ist die geschickte Kombinationsmöglichkeit aus Operation Field und der reinen Kommentarzeile. Durch Einsetzen einer Sternzeile können voneinander unabhängige Operationsangaben optisch getrennt werden. So kann der erste Step von der Jobkarte und die einzelnen Step untereinander besser differenziert werden, wie aus Abbildung 67 hervorgeht.

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
//*
//IDCAMS   EXEC PGM=IDCAMS
//SYSIN    DD *
  DELETE HLQ1.HLQ2.HLQ3
//SYSPRINT DD SYSOUT=*
//*
//IEFBR14  EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 67: Die optische Trennung von Operationsangaben

7 Das Job Statement

Ein Auftrag, und darum handelt es sich ja bei einem Job, braucht ein Identifikationsmerkmal. Diese Funktion übernimmt in der Job Control das Job Statement. Es ist das erste Statement eines Jobs und muss immer kodiert werden. Dieses erste Statement wird verwendet, um den Job zu identifizieren und dem System mitzuteilen, wie der Job verarbeitet wird.

Genau so wie der Anfang eines Auftrags spezifiziert sein muss, genau so muss auch das Ende spezifiziert sein. Hierfür gibt es allerdings unterschiedliche Möglichkeiten. Ein Job-Stream kann mehrere Jobs enthalten. Ist dies der Fall, so beginnt mit jedem Job Statement ein neuer Job. Damit ist der vorhergehende Job automatisch beendet. Enthält ein Job-Stream allerdings nur einen Job, so ist das Ende des Job-Streams auch das Ende des Jobs.

Das Job-Statement informiert das Betriebssystem über den Start eines Jobs und liefert die notwendige Abrechnungsinformation mit und gibt Laufzeitparameter an[13].

7.1 Jobname

Der „jobname“ muss immer angegeben werden. Über diesen Namen wird der Job im System identifiziert. Bei der Vergabe von „jobname“ sind folgende Regeln einzuhalten:

- „jobname“ darf bis zu **acht Stellen lang** sein
- er muss mit einem **Buchstaben oder National Character** ($, #, @) beginnen
- danach können auch **Ziffern folgen**

In den meisten Installationen gibt es feste Regeln für die Definition des Jobnamens. Deshalb müsste man sich über die entsprechenden Vorgaben informieren.

```
//jobname  job positional-parameters[,keyword-parameter]… [comments]
```

Abb. 68: Jobname

Ja nach Art und Konfiguration des Output-Facilities kann der Joblauf wie folgt angezeigt werden:

```
JOBNAME     JobID       Owner       Max-RC
JOB00001    JOB00001    USER1       CC 0000
JOB00001    JOB00002    USER1       CC 0000
JOB00001    JOB00003    USER1       CC 0000
JOB00001    JOB00004    USER1       CC 0000
JOB00001    JOB00005    USER1       CC 0000
JOB00001    JOB00006    USER1       CC 0000
```

Abb. 69: Job-Output-Facility

Nach dem Jobnamen folgt JOB als Operation Field.

[13] vgl. G.D.Brown: JCL Job Control Language im Betriebssystem OS/390 MVS, 3. Aufl., S. 73

```
//jobname  job positional-parameters[,keyword-parameter]… [comments]
```

Abb. 70: Job

Das Schlüsselwort JOB identifiziert dieses Statement als Job-Statement. Zur Abtrennung von den anderen Informationen muss vor und hinter Job mindestens je ein Blank stehen.

Als nächstes, nach dem Schlüsselwort JOB, folgen die sogenannten „positional-parameters“.

```
//jobname  job positional-parameters[,keyword-parameter]… [comments]
```

Abb. 71: positional-parameters

Diese werden durch ihre Position und Reihenfolge im Job-Statement erkannt und müssen deshalb auch immer genau in der gleichen Reihenfolge geschrieben werden. Im Job-Statement gibt es zwei positional-parameters:

- accounting
- programmers name

Die „keyword-parameter“ folgen im Anschluss an die positional-parameter.

```
//jobname  job positional-parameters[,keyword-parameter]… [comments]
```

Abb. 72: keyword-parameter

Die Reihenfolge der keyword-parameter ist beliebig, da sie über ihre Schlüsselworte identifiziert werden. Einige, wichtige keyword-parameter für den täglichen Gebrauch sind:

- CLASS
- TIME
- MSGCLASS
- MSGLEVEL
- NOTIFY
- REGION
- USER
- COND
- TYPRUN

Auf diese keyword-parameter wird später genauer eingegangen.

7.1.1 ID Field

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 73: ID Field

Jede Zeile der Job Control muss, mit je einem Kommata fortgeführt, mit zwei Schrägstrichen beginnen. Sollten die beiden Schrägstriche am Anfang vergessen werden, interpretiert das JES diese Zeile als Datenzeile.

7.1.2 Name Field

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 74: Name Field

Mit dem Jobnamen wird der Job innerhalb des MVS identifiziert.

7.1.3 Operation Field

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 75: Operation Field

Mit dem Schlüsselwort JOB wird diese Zeile dem System gegenüber als Job Statement identifiziert.

7.1.4 Positional Parameter

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 76: Positional Parameter

Als erste Parameter folgen die positional parameter. Diese müssen immer in der genau festgelegten Reihenfolge geschrieben werden.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 77: Accounting

Unter dem „accounting“ werden die Abrechnungsinformationen zum Job zusammengefasst. Die Regeln für die Angabe der Account-Nummer sind installationsabhängig.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 78: Programmers name

Scheibe ist in dieser Job Control der Name des Programmierers.

7.1.4.1 Programmer´s Name

Der Programmiername ist bis zu 20 Zeichen lang und wird meistens dazu verwendet, den Namen des Programmierers auf den Vorspann der Liste zu drucken. Es können natürlich auch andere Angaben gemacht werden, wie z.B. Telefonnummer, User-ID usw.

Es wird aber nicht nur der Name des Programmierers eingetragen. Sehr häufig wird programmer´s name auch dazu benutzt, Ablageorte für Listen oder die Telefonnummer des Programmierers festzuhalten.

Enthält der Programmiername Sonderzeichen, müssen diese in Hochkommata kodiert werden.

7.1.5 Keyword Parameter

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 79: Keyword Prameter

Nach den positionellen Parametern kommen die Schlüsselwortparameter. Die Reihenfolge dieser Parameter ist beliebig, da die Identifikation über den Namen erfolgt.

Da alle Parameter zusammen nicht in eine Zeile passen – es stehen maximal 72 Spalten zur Verfügung – wird eine Fortsetzungszeile benötigt. Fortsetzungszeilen beginnen, wie alle Zeilen der Job Control, mit zwei Schrägstrichen auf Spalte eins und zwei.

Die Fortsetzung selbst kann

- frühestens auf Spalte vier

und muss

- spätestens auf Spalte 16 beginnen.

In allen vorhergehenden Zeilen ist als letztes Zeichen ein Komma zu kodieren.

7.1.5.1 CLASS

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 80: CLASS

Der keyword parameter CLASS gibt dem Betriebssystem an, welche Job-Klasse und damit welcher Initiator benutzt werden soll. Der Initiator ist der Teil der Maschine, in den ein Job ausgeführt wird. Je nach Installation sind den verschiedenen Job-Klassen eine unterschiedliche Anzahl von Initiators zugewiesen.

Job-Klasse (CLASS)	Anzahl der Initiators
A	10
B	5
C	10
D	20
...	...
X	1
Y	1
Z	1
0	10
1	10
...	...
8	5
9	5

Tab. 7: Initiatorzuweisung pro Jobklasse

Wie aus Tabelle 7 deutlich wird, ist es für einen JCL-Programmierer wichtig zu wissen, in welcher Job-Klasse er seinen Job laufen lassen kann. Diese Information kann er sich in aller Regel von den Mitarbeiters der Systemtechnik des Rechenzentrum holen, da hier definiert wird, welche der 36 möglichen Job-Klassen verwendet werden dürfen. Im Rechenzentrum wird ebenfalls definiert, wie die Standardannahme heißt, falls die Job-Klasse im Job-Statement nicht angegeben wird.

Wenn eine größere Anzahl von JCL-Programmierern ihre Jobs in Job-Klasse Z laufen lassen wollen und alle ihre Jobs gleichzeitig starten, kann es, je nach dem, wie die einzelnen Joblaufzeiten sind, zu erheblichen Wartezeiten kommen.

Sobald ein Job gestartet wird, bekommt dieser von Betriebssystem automatisch eine sequentielle JobID zugewiesen. Stehen mehrere Jobs in der Input-Queue wird der nächste auszuführende Job anhand der nächsten

JobID ausgewählt und an den Initiator übergeben. In einer Job-Output-Facility könnte eine solche Queue wie folgt aussehen (Grobfassung):

JOBNAME	JobID	Owner	Max-RC
USER1	JOB00001	USER1	CC 0000
USER2	JOB00002	USER2	CC 0000
USER3	JOB00003	USER3	JCLERR
USER4	JOB00004	USER4	CC 0004
USER5	JOB00005	USER5	CC 0000
USER6	JOB00006	USER6	CC 0008

Abb. 81: Job-Output-Facility

Das in Abbildung 81 dargestellte Beispiel zeigt sechs unterschiedliche Jobs in Spalte Jobname von sechs unterschiedlichen JCL-Programmierern in Spalte Owner. Der Owner-Name und der Jobname sind jeweils gleich. Die Max-RCs[14], es handelt sich hier um die sogenannten Laufergebnisse, sind in den Jobs „USER1, USER2“ und „USER5“ gleich. In den Jobs „USER3, USER4“ und „USER6“ weichen die Laufergebnisse voneinander ab. Die JobID wird vom Betriebssystem nach jedem Jobstart sequentiell vergeben.

Sobald die Jobs „USER3, USER4“ und „USER6“ neu gestartet werden, erhalten diese auch vom Betriebssystem eine neue JobID. Anhand dieser JobID entscheidet nämlich das Betriebssystem, welcher Job dem nächsten freien Initiator zugewiesen wird.

JOBNAME	JobID	Owner	Max-RC
USER1	JOB00001	USER1	CC 0000
USER2	JOB00002	USER2	CC 0000
USER3	JOB00003	USER3	JCLERR
USER4	JOB00004	USER4	CC 0004
USER5	JOB00005	USER5	CC 0000
USER6	JOB00006	USER6	CC 0008
USER3	JOB00007	USER3	CC 0000
USER6	JOB00008	USER6	CC 0000
USER4	JOB00009	USER4	CC 0000

Abb. 82: Job-Output-Facility nach Job-Restart

In dem in Abbildung 82 dargestellten beispiel wurden die Jobs „USER3, USER4“ und „USER6“, dessen Max-RC von 0 abwich, in folgender Sequenz neu aufgesetzt:

1. USER3
2. USER6
3. USER4

Genau in dieser Reihenfolge hat das Betriebssystem die nächsten JobIDs vergeben, nämlich

1. JOB00007
2. JOB00008
3. JOB00009

[14] Abk. = Maximaler ReturnCode

Wenn nun die Jobklasse „Z“ für alle Jobs gewählt wurde und die Laufzeiten der einzelnen Jobs vorliegen, kann die Wartezeit/Job errechnet werden:

Jobname	JobID	Startzeit	Endezeit	Laufzeit	Wartezeit
USER1	JOB00001	12:00	12:05	5 Min.	0 Min.
USER2	JOB00002	12:05	12:15	10 Min.	5 Min.
USER3	JOB00003	12:15	12:20	5 Min.	15 Min.
USER4	JOB00004	12:20	12:30	10 Min	20 Min.
USER5	JOB00005	12:30	12:35	5 Min	30 Min.
USER6	JOB00006	12:35	12:40	5 Min.	35 Min.

Tab. 8: Job-Lauf/-Wartezeiten

Wie aus Tabelle 8 ersichtlich wird, kann selbst bei Job mit relativ kurzer Laufzeit zu erheblichen Wartezeiten kommen, wenn andere Job, mit relativ langer Laufzeit, den Initiator der Maschine belegen.

Deshalb ist es wichtig, den Job-Klassen in der Produktionsumgebung möglichst viele Initiators zuzuweisen, um die Wartezeiten möglichst auf ein Minimum zu beschränken.

7.1.5.2 MSGCLASS

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 83: MSGCLASS

Mit der MSGCLASS wird das Ausgabemedium definiert, auf dem das JOB LOG ausgegeben wird, z.B. IOF, SMS, SDSF usw. Dabei wird die Druckausgabe vom System zuerst auf eine Platte geschrieben, bevor der Output-Writer die Daten auf Papier druckt. Das JOB LOG wiederrum enthält alle Informationen über die Ausführung des Jobs.

Genauso wie die CLASS ist auch die MSGCLASS ein einzelnen Zeichen, das von A – Z oder von 0 – 9 definiert ist. Je nach Installation ist die Angabe der MSGCLASS nicht erforderlich, da hier ein Standardwert vordefiniert ist.

Werden Werte verwendet, dann darf es sich nur um solche Werte handeln, welche in der Installation festgelegt sind. Sobald andere Werte verwendet werden, werden die Informationen nie ausgegeben, da der MSGCLASS ein entsprechendes Ausgabemedium wie z.B. IOF, SMS, SDSF etc. zugeordnet werden muss.

Es folgt eine Auflistung von JCL-Statements, wie sie über den Eingabestrom eingeliefert werden. Folgende Zeichenkombinationen finden sich dabei in

den Stellen 1 bis 3[15]:

//	Normales JCL-Statement aus dem Eingabestrom
XX	JCL-Statement aus einer katalogisierten Prozedur
++	JCL-Statement aus einer Prozedur im Eingabestrom
X/	DD-Statement aus dem Eingabestrom, das ein Statement einer katalogisierten Prozedur überschreibt
+/	DD-Statement aus dem Eingabestrom, das ein Statement aus einer Prozedur aus dem Eingabestrom überschreibt
//*	JCL-Statement, das als Kommentar-Statement interpretiert wurde. Es kann sich auch um ein JES3-Statement im Eingabestrom handeln.
***	Kommentar-Statement aus dem Eingabestrom
XX*	Kommentar-Statement aus einer katalogisierten Prozedur
++*	Kommentar-Statement aus einer Prozedur im Eingabestrom

Abb. 84: MSGCLASS (JCL-Ausgabe)

7.1.5.3 NOTIFY

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 85: NOTIFY

Wenn ein Job gestartet wird, wird er in einen anderen Bereich des Speichers zur Ausführung übertragen. Das System weiß von diesem Moment an gar nicht mehr, zu welchem User der Job gehört. Ein solcher Job ist eine eigenständige Aktivität innerhalb des Systems.

Natürlich möchte der User wissen, ob und wie der Job fertig geworden ist. Diese Aufgabe übernimmt der Parameter NOTIFY im Job Statement. Der Notify Parameter legt fest, an welche User-ID die Endmeldung des Jobs geschickt werden soll.

Durch den Parameter NOTIFY wird die Endmeldung des Jobs genau an den TSO-User geschickt, der im Parameter spezifiziert ist. Das kann irgendeiner sein und muss nicht derjenige sein, der den Job abgeschickt hat.

In Abbildung 85 wurde eine Systemvariable verwendet. Diese Variable erkennt, von welchem User der Job gestartet wurde. Dadurch wird vermieden, dass jeder User seine User-ID explizit angeben muss bevor er den Job startet bzw. dass die Endmeldung des Job aus Versäumnisgründen an einen falschen User geschickt wird.

[15] vgl. G.D.Brown: JCL Job Control Language im Betriebssystem OS/390 MVS, 3. Aufl., S. 82

7.1.5.4 TIME

Der Parameter TIME legt die CPU-Zeit fest, die von einem Job höchstens verbraucht werden darf. Ist diese Zeitmenge verbraucht, bricht das Betriebssystem den Job ab. Damit wird verhindert, dass das System durch Programmfehler wie z.B. Endlosschleifen blockiert wird.

Das folgende Beispiel zeigt eine Job Control, die dem Job JOB00001 eine Laufzeit von einer Minute und 30 Sekunden zugesteht:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             TIME=(1,30),
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 86: TIME

Wenn nur Minuten angegeben werden dürfen die runden Klammern fehlen.

Das folgende Beispiel zeigt eine Job Control, die dem Job JOB00001 eine Laufzeit von 12 Minuten zugesteht:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             TIME=12,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
```

Abb. 87: TIME mit Minutenangabe

Die CPU-Zeit ist nicht identisch mit der Uhrzeit. CPU-Zeit ist generell die Zeit, die ein Job zur Ausführung von Instruktionen benötigt. Hier sind auch keine Wartezeiten, Zeiten, welche das Betriebssystem benötigt und auch keine Zeiten, die andere Jobs im Rahmen des Multiprogramming zur Ausführung von Instruktionen in Anspruch nehmen, enthalten. Prinzipiell darf der TIME-Parameter auch im EXEC-Statement kodiert werden, um einem oder mehreren Steps Zeitlimits mitzugeben. Der TIME-Parameter wird jedoch üblicherweise im Job-Statement kodiert.

7.1.5.5 MSGLEVEL

Mit dem Parameter MSGLEVEL kann der Umfang des Ausgabeprotokolls festgelegt werden. Die folgenden Beispiele zeigen die Syntax des MSGLEVEL.

```
MSGLEVEL=([statements][,messages])
```

Abb. 88: MSGLEVEL

MSGLEVEL legt den Umfang des JOB LOG fest. Je nach Angabe von statements und messages kann es stark eingeschränkt werden.

Hinweis: Zu viele Einschränkungen könne die Fehlersuche erschweren, da Fehlermeldungen zwar mit ausgegeben werden, aber durch MSGLEVEL ausgeblendet werden können.

```
MSGLEVEL=([statements][,messages])
```

Abb. 89: MSGLEVEL (statements)

Statements legt fest, welche Statements von der Job Control gedruckt werden sollen. Der Subparameter kann folgende Werte annehmen:

0 = nur das Job Statement drucken
1 = alle JCL, JES und PROC Statements, inkl. aller Auflösungen drucken
2 = nur JCL und JES Statements, so wie sie eingegeben wurden, drucken

```
MSGLEVEL=([statements][,messages])
```

Abb. 90: MSGLEVEL (messages)

Messages bestimmt den Umfang der Nachrichten des Systems zur Job Control. Der Subparameter kann folgende Werte annehmen:

0 = nur JCL Messages drucken, bei Abbruch auch JES und Operator Messages
1 = alle JCL, JES und Operator Messages drucken

Die folgende Abbildung zeigt ein Beispiel für die Angabe von MSGLEVEL Parametern:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             MSGLEVEL=(1,1),
//             NOTIFY=&SYSUID
```

Abb. 91: MSGLEVEL Parameter

Das JOB LOG wird aufgrund dieser MSGLEVEL Angabe komplett mit allen möglichen Informationen ausgegeben.

7.1.5.6 REGION

Wird ein Job gestartet, so belegt er Platz im virtuellen Speicher. Die Größe dieses Address Space ist installationsabhängig vorgegeben. Nun kann es aus verschiedenen Gründen vorkommen, dass dieser Platz nicht ausreicht, weil z.B. das Programm extrem groß ist oder sehr viele verschiedene Datenbestände benötigt werden.

In diesem Fall bietet der Parameter REGION die Möglichkeit, die Größe der Region individuell festzulegen. Benötigt ein Job mehr Address Space als die Installation standardmäßig festlegt, kann er Address Space über den REGION-Parameter erweitert werden. Der benötigte Address Space kann in

Kilobyte oder in Megabyte angegeben werden.

Der maximal zur Verfügung stehende, virtuelle Adressraum ist vom Betriebssystem abhängig. Das folgende Beispiel zeigt eine Job Control, über die der Address Space auf Kilobyte-Ebene erweitert wird:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             TIME=(1,30),
//             MSGCLASS=T,
//             MSGLEVEL=(1,1),
//             NOTIFY=&SYSUID,
//             REGION=1024K
```

Abb. 92: REGION in Kilobyte

Für den Job JOB00001 wurde ein Speicherplatz von 1024 Kilobyte reserviert. Im nun folgenden Beispiel wird Speicherplatz in Megabyte reserviert:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             TIME=(1,30),
//             MSGCLASS=T,
//             MSGLEVEL=(1,1),
//             NOTIFY=&SYSUID,
//             REGION=1M
```

Abb. 93: REGION in Megabyte

Die REGION-Angabe im Job-Statement gilt für alle nachfolgenden Steps und überschreibt etwaige Angaben, welche in einzelnen Steps gemacht wurden.

7.1.5.7 USER

Ein Job ist, wie in Unterkapitel 6.1.5.3 bereits erwähnt, eine eigenständige Aktivität innerhalb des Systems. Das System weiß nicht, welchem User der Job gehört.

Da aber auch Batchjobs unter der Kontrolle von RACF laufen und RACF für seine Schutzmaßnahmen Informationen über den Auftraggeber, der den Job abgeschickt hat, benötigt, können diese Informationen über den Parameter USER dem System bzw. RACF mitgeteilt werden.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             TIME=(1,30),
//             MSGCLASS=T,
//             MSGLEVEL=(1,1),
//             NOTIFY=&SYSUID,
//             REGION=1M,
//             USER=USER1
```

Abb. 94: USER

Auch bei diesem Parameter wird, wie beim Parameter NOTIFY, eine User-ID angegeben. Es muss eine zulässige, dem System bekannte User-ID eingetragen werden. Bei allen Aktivitäten dieses Jobs überprüft das RACF anhand dieses Parameters die Berechtigung zu diesen Arbeiten. Evtl. ist es erforderlich, zur UserID ein gültiges Passwort oder eine gültige Gruppe anzugeben.

7.1.5.8 COND

Ein Job hat in der Regel mehrere, verschiedene Steps. Die verschiedenen Steps laufen normalerweise nacheinander ab. Das nachfolgende Beispiel zeigt anhand eines Zustandsdiagrammes, wie Daten in einem Job verarbeitet werden könnten:

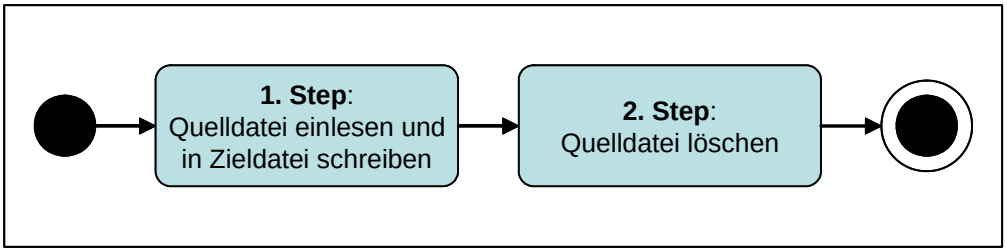

Abb. 95: Zustandsdiagramm für mögliche Jobverarbeitung

Das Löschen der Quelldatei soll aber erst dann erfolgen, wenn der Schreibvorgang erfolgreich abgeschlossen wurde. Sobald beim Schreiben in die Zieldatei ein Fehler auftritt, soll der Job beendet, d.h. der Löschvorgang der Quelldatei soll nicht durchgeführt werden.

Die Überwachung des gesamten Ablaufes kann über die Condition-Code-Abfrage erfolgen. Als Mittel dazu steht der Returncode aus dem Programm zur Verfügung. Returncodes sind Werte, die von jedem Programm ausgegeben werden und vom Betriebssystem bei der Bearbeitung des Jobs überprüft und behandelt werden.

Jeder Step kann bei seinem Ende einen Return-Code (Rückkehrkode) an das System übergeben. Der COND-Parameter steuert die Ausführung von Steps in Abhängigkeit vom Wert des Return-Codes vorangegangener Steps. Wird kein COND-Parameter angegeben findet keine entsprechende Prüfung statt.[16]

Mit dem Cond-Parameter bezieht sich der Programmierer genau auf diese Returncodes. Der Cond-Parameter dient dazu, anhand des von einem Programm gesetzten Returncodes zu entscheiden, ob die restlichen Steps innerhalb dieses Jobs ausgeführt werden sollen oder der Job beendet werden soll.

[16] vgl. G.D.Brown: JCL Job Control Language im Betriebssystem OS/390, 3. Aufl., S. 101

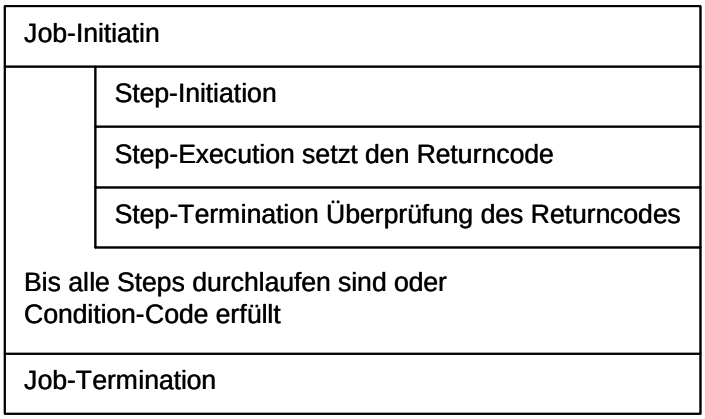

Abb. 96: Struktogramm

- nachdem die Job-Initiation abgeschlossen ist, wird die Step-Initiation des ersten Steps durchgeführt.
- daran angeschlossen wird der erste Step ausgeführt. Das in diesem Step gestartete Programm liefert zum Abschluss einen Returncode an das Betriebssystem zurück.
- In der Step-Termination wird der Returncode überprüft. Wenn die durch den COND-Parameter spezifizierte Bedingung zutrifft, wird sofort zur Job-Termination verzweigt.
- In allen anderen Fällen überprüft das System, ob noch weitere Steps vorhanden sind. Wenn ja, werden diese ausgeführt, analog zum ersten Durchlauf. Sind alle Steps abgearbeitet, verzweigt das Betriebssystem in die Job-Termination.
- Abschliessend wird in der Job-Termination der gesamte Job beendet.

Ein Anwendungsprogramm kann dem System also am Ende eines Steps einen Code übergeben. Er heißt Return-Code oder Completition-Code (Beendigungskode). Mit Hilfe von COND können Steps die Werte vorangegangener Steps auswerten. Das Betriebssystem liefert am Ende eines Jobs einen System-Completition-Code.[17]

Der System-Completition-Code ist ein 3-stelliger Wert, dem ein „S“ vorangestellt wird. Der Normalwert ist 000, wodurch normale Beendigung angezeigt wird. Falls das System oder der Systemoperator den Job abgebrochen haben, nennt der System-Completition-Code den Grund dafür.[18]

```
COND=(code,operator)
```

Abb. 97: COND (code)

“code“ ist der Wert, gegen den der Returncode des Programms bei Stepende geprüft wird.

```
COND=(code,operator)
```

Abb. 98: COND (operator)

[17] vgl. G.D.Brown: JCL Job Control Language im Betriebssystem OS/390, 3. Aufl, S. 101 ff
[18] vgl. G.D.Brown: JCL Job Control Language im Betriebssystem OS/390, 3. Aufl. S. 102

“operator“ ist die Bedingung, mit der dieser Vergleich durchgeführt wird.

Die folgende Abbildung zeigt mögliche Operatoren, mit denen Vergleiche durchführbar sind:

Operator	Bedeutung	
GT	greater than	größer als
GE	greater than or equal	größer gleich
EQ	equal to	gleich
LT	less than	kleiner als
LE	less than or equal	kleiner gleich
NE	not equal	ungleich

Tab. 9: Operatoren

Es besteht weiterhin die Möglichkeit, mehrere Bedingungen miteinander zu verknüpfen, wie die folgende Abbildung zeigt:

```
COND=((code,operator)[,(code,operator)]…)
```

Abb. 99: COND (mehrere Bedingungen)

Sollen verschiedene Bedingungen zum Jobende führen, besteht die Möglichkeit, bis zu acht verschiedene Bedingungen miteinander zu kombinieren. Sobald eine dieser Bedingungen wahr wird, wird der Job beendet.

In der folgenden Job Control wird der gesamte Job beendet, sobald einer der Steps den Returncode vier erzeugt:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             TIME=(1,30),
//             MSGCLASS=T,
//             MSGLEVEL=(1,1),
//             NOTIFY=&SYSUID,
//             REGION=1M,
//             USER=USER1,
//             COND=(4,EQ)
```

Abb. 100: COND = 4

In der folgenden Job Control wurde kodiert, dass der Job nicht beendet werden soll, solange in einem der ausgeführten Programme ein Returncode erzeugt wird, der zwischen vier und acht liegt:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             TIME=(1,30),
//             MSGCLASS=T,
//             MSGLEVEL=(1,1),
//             NOTIFY=&SYSUID,
//             REGION=1M,
//             USER=USER1,
//             COND=((4,GT),(8,LT))
```

Abb. 101: COND zwischen 4 und 8

Mit der COND-Bedingung im JOB-Statement lassen sich also die Bedingungen festlegen, unter denen ein Job beendet werden soll.

7.1.5.9 TYPRUN

Durch den Parameter TYPRUN ist es möglich, nur einzelne Aktivitäten eines Job-Ablaufes durchzuführen und andere, die nicht benötigt werden, zu unterdrücken. TYPRUN legt fest, was das JES mit der Job Control tun soll.

- TYPRUN=COPY

Diese Angabe kopiert den JCL-Stream direkt in die Ausgabe, ohne den Job zur Ausführung einzustellen. Diese Angabe kann jedoch nur in der JES2-Umgebung kodiert werden.

- TYPRUN=HOLD

Diese Angabe hält die Ausführung des Jobs zurück, bis sie vom Operator freigegeben wird. Das Einlesen der Job Control und die Conversion werden ganz normal durchgeführt. Die Execution muss allerdings explizit freigegeben werden. Ist die JCL fehlerhaft, wird der Job nicht zurückgestellt. Der Job wird dann sofort an die Outputphase zur Ausgabe weitergegeben.

- TYPRUN=JCLHOLD

Diese Angabe kann nur in der JES2-Umgebung kodiert werden. Sie veranlasst, dass das JES2 die Bearbeitung der JCL zurückhält, bis sie vom Operator freigegeben wird, d.h. hier wird ausschließlich die READER-Phase aktiv. Alle anderen Phasen laufen erst bei expliziter Freigabe ab.

- TYPRUN=SCAN

Diese Angabe prüft die JCL auf formale Fehler. Der Job wird nicht zur Ausführung eingestellt, d.h. es werden auch keine Ressourcen zugeteilt. Es werden dann Output-Phase und ein Teil der Conversion durchlaufen. Die Auflösung von Prozeduren und die Syntaxprüfung erfolgt.

Mit TYPRUN=SCAN kann allerdings nur die formale Richtigkeit der Job Control überprüft werden, d.h.:

- Schreibweise von Schlüsselwörtern
- gültige Zeichen
- paarige Klammern
- positionelle Parameter
- Syntax der JCL aus Prozeduren

SCAN kann nicht die Semantik prüfen. Unbemerkt bleiben also falsch platzierte Statements, Syntaxfehler in Subparametern sowie Parameter und Subparameter, die nicht zusammengehören. Das folgende Beispiel zeigt eine Job Control, bei der nur die formale Richtigkeit überprüft wird:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             TIME=(1,30),
//             MSGCLASS=T,
//             MSGLEVEL=(1,1),
//             NOTIFY=&SYSUID,
//             REGION=1M,
//             USER=USER1,
//             COND=(4,EQ),
//             TYPRUN=SCAN
```

Abb. 102: TYPRUN SCAN

Bei dieser Variante kommt es zu keiner Job-Execution.

8 Das EXEC-Statement

8.1 Aufbau und Verwendung

Ein Job muss mindestens einen Arbeitsablauf beinhalten. Die maximal zulässige Anzahl ist 255. Die für einen Arbeitsablauf notwendige Arbeitsanweisung ist das EXEC-Statement. EXEC steht für EXECute, was soviel bedeutet wie ausführen.

- Jedes EXEC-Statement ist der Beginn eines Steps im Job.

Das EXEC-Statement benennt ein Programm oder eine Prozedur, die an dieser Stelle im Jobablauf ausgeführt werden soll und liefert dem System Angaben zur Ausführung. Das folgende Beispiel zeigt den Syntax-Aufbau eines EXEC-Statements:

```
//[stepname] EXEC positional-parameter[,keyword-parameter]…[comments]
```

Abb. 103: Syntax-Aufbau eines EXEC-Statements

Das folgende Beispiel zeigt eine komplette Job Control mit dem Syntax-Aufbau eines EXEC Statements:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T
//*
//STEP1    EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            DSORG=PS,
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 104: EXEC-Statement in der Job Control

Man beachte in Abbildung 104 auch die Zeichenkette „//“ im ID-Field sowie den „*“ im Name-Field zur Erstellung einer Kommentarzeile, um mit deren Hilfe das EXEC-Statement optisch vom JOB-Statement abzugrenzen.

8.2 Positional Parameter

Über den Positional Parameter wird die Arbeit festgelegt. Diese auszuführende Arbeit kann sein:

- ein einzelner Arbeitsschritt, also ein Programm
- eine Abfolge von Arbeitsschritten, die in einem Unterauftrag zusammengefasst sind, eine sogenannte Prozedur.

Für die Syntax des positionellen Parameters gibt es somit zwei Formen, die nach Programmaufruf und nach Prozeduraufruf unterschieden werden.

8.2.1 Programmaufruf und Programmname

```
//STEP1    EXEC PGM=program-name
```

Abb. 105: Programmaufruf

“program-name“ ist der Name eines Members in eine Lademodulbibliothek, d.h. der Name eines ausführbaren Programms.

Programme können sich in einer System-, einer temporären oder einer privaten Bibliothek befinden. Ein Programm ist ein Load Module, das als Member eines Partitioned Datasets auf einer Platte gespeichert ist. Wird ein Programm beim Aufruf nicht gefunden, wird der Job vom System beendet und der ABEND-Code S806 ausgegeben.

8.2.1.1 Programme in der Systembibliothek

Die Systembibliothek „SYS1.LINKLIB“ enthält alle von der IBM bereitgestellten Systemprogramme. Dazu gehören die Compiler, der Linkage Editor und die Utilities. Diese Bibliothek kann auch hauseigene Programme aufnehmen, solange die Namen nicht mit den Namen der Systemprogramme von der IBM übereinstimmen.

Wird dennoch ein Programm in die Systembibliothek geladen, dessen Namen dort bereits existiert, besteht die Gefahr, dass das bereits vorhandene Systemprogramm unwiederruflich überschrieben wird. Es ist also ratsam, sowohl eine regelmäßige Sicherung der Systembibliothek zu ziehen also auch den Programmierern generell zu untersagen, ihre fertigen Programme auf die Systembibliothek zu stellen. Evtl. sollte hierfür eine eigene Bibliothek eingerichtet werden.

8.2.1.2 Programme in privaten Bibliotheken

Die Programme müssen als Member einer Bibliothek eindeutige Namen tragen. Programme in verschiedenen Bibliotheken können natürlich gleich benannt sein. Es sind jedoch Konsequenzen bei der Angabe einer JOB- bzw. STEPLIB zu berücksichtigen, die nicht ganz unerheblich sind.

So können nicht standartisierte Bibliotheken, also diejenigen, die nicht automatisch vom System während eines Joblaufes angesprochen werden, über eine JOB- oder/und STEPLIB-Definition verknüpft werden.

8.2.1.2.1 JOBLIB

JOBLIB ist ein DD-Statement zur Angabe einer Verknüpfung von Programm-Lade-Bibliotheken, die nicht automatisch während eines Joblaufes vom System angesprochen werden, weil diese nicht in den Systemeinstellungen hinterlegt und somit dem System gänzlich unbekannt sind.

Eine JOBLIB-Angabe wird für einen kompletten Job definiert und kann nur durch eine STEPLIB-Angabe stepweise substituiert werden. Die Angabe

einer JOBLIB kann folgendermaßen aussehen:

```
…
//JOBLIB   DD DSN=HLQ1.HLQ2,
//            DISP=SHR
//         DD DSN=HLQ3.HLQ4.HLQ5,
//            DISP=SHR
//         DD DSN=HLQ6.HLQ7,
//            DISP=SHR
…
```

Abb. 106: JOBLIB

In Abbildung 106 wurden drei Datasets definiert, in welchen das System nach Programmen suchen kann, die nicht auf den in den Standardeinstellungen bekannt gegebenen Bibliotheken stehen.

Während eines Joblaufes werden alle drei Datasets sequentiell abgearbeitet, solange bis das System das im Step aufgerufene Programm gefunden hat. Findet das System das Programm nicht wird der Job mit ABEND-Code S806 beendet.

Befinden sich jedoch zwei unterschiedliche Versionen eines Programmes auf zwei unterschiedlichen Bibliotheken kann es durchaus passieren, dass die falsche Version aktiviert wird.

8.2.1.2.2 STEPLIB

STEPLIB ist auch ein DD-Statement und ähnelt dem JOBLIB-DD-Statement. Es ist allerdings auf Stepebene angesiedelt. Es muss hinter einem EXEC-Statement stehen und gilt nur für diesen Step. Durch die Angabe eines STEPLIB-DD-Statements werden JOBLIB-Angaben ungültig, jedoch nur für den Step, in welchem die STEPLIB-Angabe gemacht wurde.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T
//*
//STEP1    EXEC PGM=IEFBR14
//STEPLIB  DD DSN=SYS1.IBM.LOADLIB,
//            DISP=SHR
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            DSORG=PS,
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 107: STEPLIB

8.2.1.2.3 JOBLIB und STEPLIB in einer JCL

Das folgende Beispiel zeigt, wie eine JCL aussehen könnte, welche JOBLIB- und STEPLIB-DD-Statements enthält:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T
//*
//JOBLIB   DD DSN=HLQ1.HLQ2,
//            DISP=SHR
//         DD DSN=HLQ3.HLQ4.HLQ5,
//            DISP=SHR
//         DD DSN=HLQ6.HLQ7,
//            DISP=SHR
//*
//STEP1    EXEC PGM=HUGO
//STEPLIB  DD DSN=SYS1.IBM.LOADLIB,
//            DISP=SHR
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            DSORG=PS,
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 108: JOBLIB und STEPLIB in einer JCL

In Abbildung 108 wird ein JOBLIB-DD-Statement eingesetzt, um nicht standartisierte Lademodul-Bibiliotheken anzusprechen. Das Programm IEFBR14 befindet sich jedoch in einer anderen Bibliothek, welche nur für diesen Step angesprochen werden soll. Durch den Einsatz des STEPLIB-DD-Statements wird diese Anforderung realisiert und das JOBLIB-DD-Statement für diesen Step ignoriert.

Hinweis: Empirische Tests haben gezeigt, dass Standard-Modul-Lade-Bibliotheken auch bei JOBLIB- und STEPLIB-Angaben angesprochen werden. Es kann sich hierbei jedoch auch um Einstellungen handeln, die zwischen den einzelnen Installationen variieren können. Desweiteren wird in den Kapiteln 8.2.1.2.1 und 8.2.1.2.3 von Bibliotheken gesprochen. Es kann sich jedoch auch um sogenannte PO-Dateien handeln.

Auch in dieser Abbildung wird der Vorteil deutlich, wenn die Zeichenkette „//“ im ID-Field und der „*“ im Name-Field zur Erzeugung einer Kommentarzeile eingesetzt wird, um eine optische Abgrenzung der einzelnen Job-Segmente

1. JOB-Statement
2. JOBLIB-DD-Statement
3. EXEC-Statement

zu erreichen.

8.2.2 Prozeduraufruf und Prozedurname

```
//STEP1    EXEC PROC=procedure-name
```

Abb. 109: Prozeduraufruf

Eine Prozedur kann auch wie folgt aufgerufen werden:

```
//STEP1    EXEC procedure-name
```

Abb. 110: Prozeduraufruf ohne PROC-Angabe

Im Gegensatz zum Aufruf eines Programms ist beim Aufruf einer Prozedur die Angabe „PROC“ nicht erforderlich. Wenn der Programmierer die Angabe „EXEC PGM=“ nicht macht sondern sich nur auf die Angabe „EXEC „, wie in Abbildung 110 verdeutlicht, beschränkt, unterstellt das System, dass hier eine Prozedur aufgerufen werden soll.

“procedure-name“ ist der Name eines Members in einer Prozedurbibliothek. Ein Job kann bis zu 255 EXEC Statements enthalten. Mehr sind nicht erlaubt. Die EXEC´s in den aufgerufenen Prozeduren werden hier mitgezählt.

8.2.2.1 JCLLIB

Mit Hilfe einer JCLLIB-Angabe kann der Programmierer Prozedur-Bibliotheken aufrufen, die vom System standardmäßig nicht erkannt werden, also im Prinzip das gleiche wie beim JOBLIB- und STEPLIB-DD-Statement. Der Unterschied liegt nun darin, dass bei der JCLLIB-Angabe keine Programme, sondern JCL-Prozeduren aufgerufen werden.

```
…
//PROCLIB  JCLLIB ORDER=(SYSTEM.PROCLIB)
…
```

Abb. 111: JCLLIB

Wie aus den Abbildungen 106, 107 und 108 ersichtlich wird, sind die Bezeichnungen „JOBLIB“ und „STEPLIB“ im Name-Field fest vordefiniert. Bei der JCLLIB-Angabe ist die Vorgabe jedoch nicht im Name-Field, sondern im Operation-Field durch die Angabe „JCLLIB“ festgelegt. Die Bezeichnung PROCLIB im Name-Field kann also variieren. Es wird auch ersichtlich, dass es sich bei der JCLLIB-Angabe nicht um eine DD-Statement-Angabe handelt.

8.3 Keyword Parameter

Mit diesem Parameter können weitere Differenzierungen individuell für den einzelnen Arbeitsablauf angegeben werden.

8.3.1 PARM

Häufig ist es wünschenswert, dass Programme bei jedem neuen Aufruf individuell gesteuert werden. Dazu muss eine Steuerungsinformation beim Aufruf an das Programm weitergegeben werden. Diese Aufgabe übernimmt der PARM Parameter.

Der Wert, der mit PARM an das Programm übergeben wird, kann von diesem verarbeitet werden. Voraussetzung dafür ist, dass das Programm entsprechend geschrieben ist.

Das folgende Beispiel zeigt die allgemeine Syntax des PARM Parameter:

```
PARM[.procstepname]=information
```

Abb. 112: Allgemeiner PARM-Parameter

Der Parameter PARM übergibt eine Zeichenkette an das auszuführende Programm. Der Inhalt der Zeichenkette wird von diesem Programm verarbeitet. Die Zeichenkette, die übergeben werden soll, muss in Hochkommata oder runde Klammern eingeschlossen werden, wenn sie selbst Kommata enthält.

Die folgenden Beispiele zeigen zwei Job Control-Varianten, die den Parameter PARM enthalten. In der ersten wurde die zu übergebende Zeichenkette in runder Klammer, in der zweiten wurde sie in Hochkommata geschrieben:

```
//STEP0001 EXEC PGM=TEST,PARM=(1989,ORDER,4711)
```

Abb. 113: PARM-Parameter in Klammern

oder

```
//STEP0001 EXEC PGM=TEST,PARM='1989,ORDER,4711'
```

Abb. 114: PARM-Parameter in Hochkommata

Würde in diesem Beispiel keine Klammer oder kein Hochkommata gesetzt werden, wäre der Parameter beim ersten Komma beendet.

8.3.1.1 Procstepname

’procstepname’ wird benötigt, wenn PARM beim Aufruf einer Prozedur mitgegeben wird, die mehr als einen Step enthält. Der Step in der Prozedur, der die Information erhalten soll, muss in diesem Fall mit „procstepname“ adressiert werden.

8.3.1.2 Information

’information’ kann maximal 100 Zeichen lang sein und enthält die zu übergebende Zeichenkette. Sonderzeichen werden in Hochkommata gesetzt. Die Hochkommata stehen dann nicht mehr als Rahmen für ‚information’ zur Verfügung. ‚information’ muss dann immer in runden Klammern stehen. Hochkommata und Ampersand müssen doppelt geschrieben werden, wenn sie Bestandteil von ‚information’ sind, da sie ansonsten als Steuerungsinformation interpretiert würden.

- Die Struktur der Zeichenkette ist durch das Programm fest vorgegeben

```
//STEP0001 EXEC PGM=TEST,PARM='1989,ORDER,'Meier + Sohn',4711'
```

Abb. 115: PARM-Parameter in doppelt geschriebenem Hochkommata

8.3.2 COND

Ein weiterer Parameter ist der COND Parameter, auch COND-Bedingung genannt. Der COND Parameter hat im EXEC Statement eine ähnliche Funktion wie im JOB Statement. Er führt hier zum Überspringen eines Steps, wenn die Bedingung erfüllt ist.

Jeder Step kann bei seinem Ende einen Return Code an das System übergeben. Der COND Parameter steuert die Ausführung von Steps in Abhängigkeit vom Wert des Return Codes vorangegangener Steps. Wenn beispielsweise eine Compilation fehlerhaft ist, macht es keinen Sinn, die Steps mit dem Lankage Editor und der Programmausführung zu aktivieren. Fehlt der COND Parameter, so findet keine entsprechende Prüfung statt und die Steps werden regulär ausgeführt. Das System ignoriert COND im ersten EXEC Statement eines Jobs, weil von vorangegangenen Steps keine Information vorliegen kann[19].

Eine andere Möglichkeit für die bedingte Ausführung von Steps ist die IF/THEN/ELSE-Bedingung. Zu beachten ist hier jedoch, dass die Logik einer COND-Bedingung anders lautet als die der IF/THEN/ELSE-Bedingung, auf welche in Kapitel 7.3.3 genauer eingegangen wird.

Beim JOB Statement kann mit dem COND Parameter, sobald die Bedingung in einem der zahlreichen Steps zutrifft, der ganze Job beendet werden. Wird der COND Parameter im EXEC Statement eingebaut, wird ein einziger Step nicht zur Ausführung gebracht, wenn die Bedingung zutrifft, d.h. der Step, bei dem die Bedingung zutrifft, wird übersprungen.

Bei dieser Steuerungsvariante kann man zwischen folgenden Optionen wählen:

1. Die COND-Bedingung soll sich auf einen bestimmten Step beziehen.
2. Die COND-Bedingung soll sich auf einen beliebigen Step beziehen.

Mit Hilfe der 1. Option kann man die Steps individueller aufeinander abstimmen und dadurch Job-Abbrüche wie JCL-Error o.ä. besser vorbeugen. Die 2. Option ist allgemeiner und kommt der COND Bedingung im JOB Statement am nächsten[20].

[19] Vgl. G.D.Brown: JCL Job Control Language im Betriebssystem OS/390 MVS, 3. Aufl., S. 101
[20] Vgl. Kapitel 6.1.5.8

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IDCAMS
//SYSIN    DD *
  LISTCAT ENTRY(HLQ1.HLQ2.HLQ3)
//SYSPRINT DD SYSOUT=*
//*
//STEP2    EXEC PGM=IEFBR14,
//            COND=(0,EQ,STEP1)
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(TRK,(1,10000),RLSE)
```

Abb. 116: COND-Bedingung im Step mit Stepbezug

Abbildung 116 zeigt ein Beispiel, in dem innerhalb einer Job Control geprüft wird, ob eine Datei vorhanden ist oder nicht. Ist sie vorhanden, erzeugt das Programm IDCAMS einen RC 0 und der zweite Step wird nicht ausgeführt. Ist sie dagegen nicht vorhanden, erzeugt IDCAMS einen RC 4 und der zweite Step wird ausgeführt.

Das folgende Zustandsdiagramm verdeutlicht diesen technischen Ablauf:

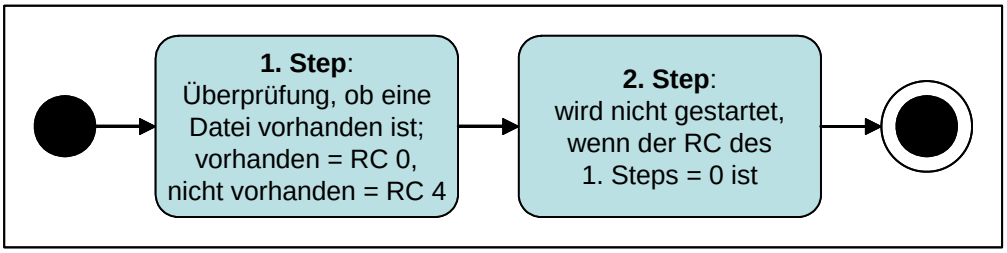

Abb. 117: Zustandsdiagramm für stepbezogene COND-Bedingung

Die allgemeine Syntax des COND-Parameters baut sich also wie folgt auf:

```
COND=(code,operator)
```

Abb. 118: COND[21]

'code' ist der Wert, gegen den der Returncode des Programms bei Stepende abgeprüft wird.

'operator' ist die Bedingung, mit der dieser Vergleich durchgeführt wird.

Immer dann, wenn die Bedingung wahr ist, wird der Step, in dem der COND Parameter angegeben wurde, nicht ausgeführt. Es wird hier die Voraussetzung spezifiziert, unter der der Step übersprungen wird. In diesem Sinne wird also das nicht Ausführen des Step als die positive Antwort definiert.

[21] vgl. Kapitel 6.1.5.8

In dieser Form der Condition Code-Abfrage auf Stepebene werden die Returncodes aller bisher gelaufenen Steps dieses Jobs überprüft.

```
COND[.procstepname]=((code,operator[,stepname][.procstepname])…)
```

Abb. 119: COND mit Bezug auf Procsteps

Es gibt aber Situationen, in denen man sich mit der Ablaufsteuerung nur auf einen Step beziehen will. Mit ‚stepname' und ‚procstepname' kann man sich explizit auf den Returncode eines Steps beziehen. Voraussetzung ist allerdings, dass der Step, auf den man sich bezieht, im Job vor dieser Condition Code-Abfrage liegt.

Zusätzlich gibt es die Möglichkeit, mehrere Bedingungen miteinander zu verknüpfen. Auch im EXEC Statement können bis zu acht Condition Code-Abfragen beliebig miteinander kombiniert werden.

Das folgende Beispiel zeigt eine Condition Code-Abfrage, bei der es keine Rolle spielt, in welchem Step der Returncode geliefert wird. Da die Abfrage immer eine Oder-Abfrage ist, reicht das Eintreten der Bedingung in einem dieser Steps aus.

```
…
//STEP1    EXEC PGM=TEST1
//*
//STEP2    EXEC PGM=TEST2
//*
//STEP3    EXEC PGM=TEST3,
//             COND=(0,LT)
…
```

Abb. 120: COND-Bedingung ohne Stepbezug

Step1 läuft in jedem Fall, da es der erste Step dieses Jobs ist. Da für Step2 keine Condition Code-Abfrage spezifiziert worden ist, wird auch dieser Step laufen. In Step3 lautet die Abfrage:

Ist 0 < als der Returncode von Step1 oder ist o < als der Returncode von Step2? Immer wenn diese Bedingung zutrifft, wird der Step3 übersprungen.

Das folgende Beispiel zeigt eine Condition Code-Abfrage, bei der die Condition Code-Abfrage speziell auf einen oder mehrere Steps ausgerichtet ist:

```
…
//STEP1    EXEC PGM=TEST1
//*
//STEP2    EXEC PGM=TEST2
//*
//STEP3    EXEC PGM=TEST3,
//             COND=((0,LT,STEP1),(4,EQ,STEP2))
…
```

Abb. 121: COND-Bedingung im Step mit mehreren Stepbezügen

Step1 läuft in jedem Fall, da es der erste Step dieses Jobs ist. Für Step2 wurde keine Condition Code-Abfrage spezifiziert, daher wird auch dieser Step laufen. In Step3 lautet die Abfrage:

Ist 0 < als der Returncode von Step1 oder ist 4 = dem Returncode von Step2? Sobald eine dieser Bedingungen zutrifft, wird der Step3 übersprungen.

Hinweis: Wie bereits erwähnt kann man den COND Parameter also sowohl im JOB Statement als auch im EXEC Statement kodieren. Es ist nur darauf zu achten, dass der COND Parameter im JOB Statement die höhere Priorität hat.

8.3.2.1 System Completition Codes

Der System Completition Code ist ein alphanumerischer, dreistelliger Wert, dem ein “S“ vorangestellt wird. Wenn ein Job normal beendet wird, lautet der Wert des System Completition Code „000“.

Sobald ein Job vom System bzw. vom Operator abgebrochen wird, nennt das System mittels System Completition Code den Grund für den Abbruch. Im Systemumfeld wird hier von einem Abnormal Ended, kurz Abend, gesprochen.

In der folgenden Tabelle werden die am häufigsten vorkommenden Abend-Codes erläutert:

Code	**Bedeutung**	**Ursache**
S0C1	Ungültige Instruktion	Möglicherweise liegt eine fehlerhafte Verzweigung in einen Datenbereich vor oder ein Programm wurde versehentlich zerstört
S0C4	Versuchter Zugriff auf einen geschützten Programm-Bereich	Möglicherweise wurde ein Index innerhalb eines Feldes falsch initiiert oder es wurde versucht, auf einen Speicherbereich zuzugreifen, der einem anderen Programm zugeordnet ist.
S0C5	Ungültige Adresse	Falsche Verzweigung oder falscher Index
S0C7	Ungültige Daten	Möglicherweise wurden Daten in falschen Strukturen gespeichert oder sind zerstört worden
S122	Der Job wurde vom Operator eliminiert	
S222	Der Job wurde vom TSO-Benutzer eliminiert	
S322	Das CPU-Limit wurde überschritten	Möglicherweise enthält das Programm eine Endlosschleife. Erhöhen Sie ggf. den TIME-Parameter im Job- bzw. EXEC-Statement
S722	Das Ausgabe-Limit wurde überschritten	
S804	Es wurde mehr Speicher benötigt als über den REGION-Parameter angegeben wurde	
S806	Das Programm, das ausgeführt werden sollte, konnte nicht	Möglicherweise ist die JOBLIB- bzw. STEPLIB-Angabe fehlerhaft oder der Programmname falsch geschrieben

	gefunden werden	
S80A	Siehe S804	
S822	Die angeforderte REGION-Größe ist nicht verfügbar.	
SB37	SPACE-Angabe zu klein definiert	Für diesen Vorgang ist es empfehlenswert, die Speichergröße der Ausgabedatei entsprechend der Eingabedatei zu definieren.
SE37	Der reservierte Speicherplatz ist zu klein	Die Datei kann nur mit 'cancel' verlassen werden. Entweder wird die Datei komprimiert oder sie muss komplett neu angelegt und der Inhalt der alten Datei kopiert werden

Tab.: 10: Abend-Codes

8.3.2.2 Return Codes

Während sich der System Completition Code auf das Ergebnis eines kompletten Joblaufes bezieht, bezieht sich der Return Code auf das Ergebnis eines Steplaufes. Man könnte sagen, der System Completition Code basiert auf dem Return Code.

Der Return Code (RC) ist ein zweistelliger, numerischer Wert und wird vom dem Programm erzeugt, das im EXEC-Statement aufgerufen wird. Die Bedeutung des ausgegebenen Return Codes ist in der Dokumentation des jeweiligen Programmes nachzulesen. Man sollte sich nicht darauf verlassen, dass der RC 4 aus Programm A die gleiche Bedeutung hat wie der gleiche RC aus Programm B.

8.3.3 IF/THEN/ELSE/ENDIF Statement Construct

Das IF/THEN/ELSE/ENDIF-Statement-Construct bietet eine wirklich einfache Hilfe, um Steps gezielt zu steuern. Man kann damit relativ einfach den kompliziert zu benutzenden COND-Parameter ablösen[22].

Wenn ein Step nur unter bestimmten Bedingungen laufen soll, können hierzu die IF/THEN/ELSE/ENDIF Statement Einheiten verwendet werden. Durch die Anwendung dieser Statement-Einheiten besteht unter anderem die Möglichkeit, Datenbestände bedingungsabhängig zuzuordnen.

Das folgende Beispiel zeigt die allgemeine Syntax der IF/THEN/ELSE/ENDIF Statement-Einheit:

```
//[name]    IF (Ausdruck) THEN [Kommentar]
//[name]    ELSE [Kommentar]
//[name]    ENDIF [Kommentar]
```

Abb. 122: Allgemeine IF/THEN/ELSE/ENDIF Statement-Einheit

Wie jedes andere Statement beginnt auch das IF Statement mit zwei Schrägstrichen im ID-Field. Ein Name kann beliebig angegeben werden, er sollte jedoch eindeutig sein. Wird kein Name angegeben, so muss die dritte Spalte frei bleiben.

[22] vgl. G.D.Brown: JCL Job Control Language im Betriebssystem OS/390 MVS, 3. Aufl., S. 109

Der Name darf bis zu acht Zeichen lang sein. Er muss mit einem Buchstaben oder einem National Character ($, #, @) beginnen. Danach können auch Ziffern folgen.

Das Operation-Field enthält das Schlüsselwort ‚IF'. Vor und hinter dem Schlüsselwort ‚IF' ist jeweils ein Leerzeichen einzusetzen. An dieser Stelle wird die Bedingung für die Ausführung der nachfolgenden Statements formuliert. Dafür stehen vergleichende, logische und verneinte Operatoren zur Verfügung. Der Ausdruck kann auch verschiedene Schlüsselwörter enthalten.

Das IF Statement wird beendet mit dem Ausdruck THEN, es sei denn, es soll noch ein Kommentar angefügt werden.

Ist die Bedingung im IF/THEN Statement falsch, so werden die JCL Statements durchlaufen, die nach dem ELSE Statement angegeben wurden. Nur wenn diese Alternative gewünscht wird, wird ein ELSE Statement kodiert.

Am Schluss der IF/THEN/ELSE-Einheit ist das ENDIF Statement. Es ist ähnlich einfach zu kodieren wie das ELSE Statement. Im IF Statement können für die Formulierung der Bedingung folgende Schlüsselwörter verwendet werden:

RC	gibt einen Returncode an
ABEND	gibt eine auftretende Abend-Bedingung an
<> ABEND	gibt an, dass keine Abend-Bedingung auftritt
ABENDCC	gibt einen System oder User Completition Code an
RUN	gibt an, dass ein bestimmter Step in Ausführung ist
<> RUN	gibt an, dass ein bestimmter Step nicht in Ausführung ist

Tab. 11: Bedingungen für die Formulierung von IF Statements

Im IF Statement können für diese Formulierung der Bedingung diese Schlüsselwörter verwendet werden. Wenn ein Returncode für den Ausdruck im IF Statement verwendet werden soll, werden noch Operatoren benötigt. Nachfolgende Abbildung zeigt eine Liste der Vergleichsoperatoren:

Operator	Bedeutung	
GT (>)	greater then	größer als
GE (>=)	greater than or equal to	größer gleich
EQ (=)	equal	gleich
LT (<)	less than	kleiner als
LE (<=)	less than or equal to	kleiner gleich
ne (<>)	not equal	ungleich

Tab. 12: Vergleichsoperatoren

8.3.4 Logische Operatoren

Um verschiedene Abfragen miteinander verknüpfen zu können werden sogenannte logische Operatoren benötigt. Diese lauten wie folgt:

- AND (&)
- OR (|)

Der AND Operator besagt, dass alle spezifizierten Ausdrücke wahr sein müssen um die Bedingung zu erfüllen, z.B.:

- Die Bedingung (RC > 8 & RC < 24) überprüft, ob der Returncode größer acht und kleiner 24 ist. Die Bedingung ist also nur dann wahr, wenn ein Returncode zwischen neun und 23 geliefert wird.

Der OR Operator legt fest, dass nur einer der Ausdrücke zutreffen muss, z.B.:

- Die Bedingung (RC = 8 | RC = 10 | RC > 24) überprüft, ob der Returncode = 8 oder = 10 oder > 24 ist. Die Bedingung ist also immer dann wahr, wenn ein Returncode von acht, zehn oder > 24 geliefert wird.

8.3.5 Der NOT Operator

Mit dem NOT Operator kann ein Ausdruck verneint werden. NOT hat die höchste Priorität, es folgen die Vergleichsoperatoren. Die logischen Operatoren AND und OR haben die niedrigste Priorität. Die folgenden beiden Statements bewirken das Gleiche:

```
…
//TEST1    IF NOT (RC > 8) THEN
…
```

```
…
//TEST1    IF (RC <= 8) THEN
…
```

Abb. 123: Zwei unterschiedliche Operatoren mit gleicher Bedeutung

Wenn der NOT Operator angegeben wird, werden sowohl der vergleichende als auch der logische Operator verneint. Einige wichtige Anmerkungen zum sinnvollen Einsatz der IF/THEN/ELSE/ENDIF-Statement Einheit:

1. Die IF/THEN/ELSE/ENDIF-Statement Einheit wird benutzt, um bedingungsabhängig verschiedene Programme aufzurufen oder alternativ Daten zu übergeben.
2. Es können bis zu 15 solcher IF/THEN/ELSE/ENDIF Einheiten verschachtelt werden. Die Steps eines THEN oder ELSE Falles können also wieder eine IF/THEN/ELSE/ENDIF Einheit enthalten.
3. Eine IF/THEN/ELSE/ENDIF-Einheit wird immer nach einem JOB Statement platziert.
4. Die Statementfolge nach einem THEN oder ELSE Statement kann leer sein.
5. Wurde ein COND Parameter im EXEC Statement kodiert, so führt das System den Jobstep nur aus, wenn beide Bedingungen erfüllt sind.

Das folgende Beispiel zeigt die Anwendung der IF/THEN/ELSE/ENDIF-Statement Einheit:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=LOHN
//*
//IFTEST   IF (RC > 4 & RC < 8) THEN
//STEP2    EXEC PGM=TEST1
//*
//         ELSE
//ERROR    EXEC PGM=TEST2
//*
//IFENDE   ENDIF
```

Abb. 124: Anwendung der IF-THEN-ELSE-ENDIF-Einheit in der JCL

Die hier kodierte Statement-Einheit legt fest, dass Step2 nur dann läuft, wenn die Bedingung im IF Statement wahr ist. Andernfalls wird der Step mit namen ERROR gestartet.

Die Bedingung im IF Statement enthält sowohl vergleichende Operatoren als auch einen logischen Operator. Die Bedingung ist wahr, wenn das Programm LOHN einen Returncode liefert, der größer vier und kleiner 8 ist.

Die Operationen können sowohl in Mnemonischer als auch in Zeichenform geschrieben werden. Werden Mnemonische Abkürzungen verwendet, so dürfen die davor und die dahinter stehende Leerstelle nicht vergessen werden. Werden Zeichen eingesetzt, können die Leerstellen entfallen.

Die logischen Operatoren AND (&) und OR (|) bilden allerdings Ausnahmen und müssen durch Leerstellen eingefasst werden. Um ganz sicher zu gehen ist es empfehlenswert, man verwendet immer Leerstellen um die logischen Operatoren herum.

Mnemonischer-Operator	Zeichen-Operator	Beschreibung
NOT	¬	Logisch nicht
EQ	=	Gleich
NE	¬=	Nicht gleich
GT	>	Größer als
LT	<	Kleiner als
GE	>=	Größer gleich
NL	¬<	Nicht kleiner (wie GE)
LE	<=	Kleiner gleich
NG	¬>	Nicht größer als
AND	&	Logisches UND
OR	|	Logisches ODER

Tab. 13: Vergleichsoperatoren für IF/THEN/ELSE/ENDIF

9 Das DD Statement

Mit Hilfe von DD-Statements werden die Daten für ein Programm beschrieben. OS/390 bietet eine Vielzahl von unterschiedlichen Datenstrukturen an.

9.1 Das Format des DD-Statements

Die allgemeine Schreibweise des DD-Statements sieht wie folgt aus:

```
//ddname   DD parameter
```

Abb. 125: Allgemeine Schreibweise des DD-Statements

Der 'ddname' ist der Name des DD-Statements und stellt die Verbindung zum Data-Control-Block (DCB) im Programm her. Es gibt drei Positionsparameter und eine Reihe von Keyword-Parametern. Die Positionsparameter lauten im einzelnen:

- '*' zeigt an, dass eine oder mehrere Datenzeilen unmittelbar hinter diesem DD-Statement folgen.
- 'DUMMY' gibt dem Dataset den Status eines Platzhalters.
- 'DATA' in Verbindung mit 'DLM=%%' zeigt an, dass Datenzeilen mit '//' bzw. '/*' in den Spalten eins und zwei unmittelbar hinter diesem DD-Statement folgen.

Die folgende Tabelle erläutert die am häufigsten benutzten Keyword-Parameter. Die Anordnung ist an keine Vorgaben gebunden.

Begriff in der JCL-Sprache	Bedeutung	Erläuterung
DSN	Data Set Name	Name einer Datei, die angesprochen werden soll (siehe Kapitel 8.3.1)
DISP	Disposition	Gibt an, wie eine Datei angesprochen wird. Mögliche Varianten: SHR, OLD, MOD (siehe Kapitel 8.3.2)
DSORG	Dataset-Organisation	Gibt an, wie eine Datei organisiert werden soll. Am häufigsten verwendet: PS, PO (siehe Kapitel 8.3.3)
RECFM	Record-Format	Gibt an, ob eine Datei ein fixes oder ein variables Satzformat haben soll (siehe Kapitel 8.3.4)
LRECL	Logische Record-Länge	Gibt die Satzlänge einer Datei an (siehe Kapitel 8.3.5)
DCB	Data Control Block	Unter dieser Angabe können LRECL und RECFM konsolidiert werden (siehe Kapitel 8.3.6)
UNIT		Benennt eine Ein-/Ausgabe-Einheit
SPACE		Reserviert die angegebene Größe an Speicherplatz
EXPDT	Expiration Date	Gibt ein Ablaufdatum für eine Datei an
RETPD	Retantion Period	Gibt eine Ablaufperiode für eine Datei an

Tab. 14: Keyword-Parameter im DD-Statement

9.2 Aufbau und Verwendung

Zu jedem Arbeitsablauf müssen die benötigten Arbeitsmittel auch im Detail spezifiziert werden. Das dafür notwendige Statement ist in der Job Control das DD-Statement. Das Data Definition Statement beschreibt einen Datenbestand und gibt die Ressourcen an, die für die Ein- und Ausgabe benötigt werden.

Job-Spezifikation (JOB-Statement)
Arbeitsablauf (EXEC-Statement)
Arbeitsmittel (//…DD…)
Arbeitsablauf (EXEC-Statement)
Arbeitsmittel (//…DD…)

Abb. 126: Einsatz von DD-Statements

Aus Programmen heraus werden Datenbestände angesprochen. Hierfür gibt es

in jeder Programmiersprache Befehle für das Einlesen und andere Befehle für das Auslesen von Daten.

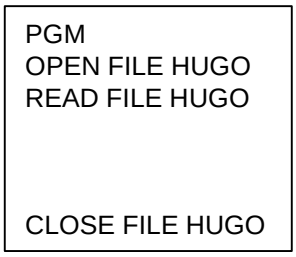

Abb. 127: Logischer Ablauf eines Programms

Durch den Befehl OPEN innerhalb des Programms wird ein Puffer angelegt. Der Puffer ist ein Bereich im eigenen Address Space. Die Löschung erfolgt durch den CLOSE-Befehl.

Durch den READ-Befehl wird aus dem Programm heraus der Puffer, hier mit dem Namen HUGO, angesprochen. Mit jedem READ-Befehl wird ein einzelner Satz aus dem Puffer in den Bereich des Programms übertragen.

Das Betriebssystem versorgt diesen Puffer regelmäßig mit Datenblöcken von externen Datenträgern. Offen ist jetzt nur noch die Frage, welche Datenbestände von welchem Datenträger dafür benutzt werden. Genau diese Aufgabe übernimmt das DD-Statement.

Über den DD-Namen wird der Name des Puffers angegeben und über den Datasetname, kurz DSN, wird spezifiziert, mit welchem Datenbestand das Betriebssystem diesen Puffer verbinden soll. Damit wird klar, dass zu jedem Datenbestand, der aus einem Programm angesprochen wird, auch ein entsprechendes DD-Statement vorhanden sein muss.

Für jeden Datenbestand, der aus einem Programm angesprochen wird, muss ein DD-Statement kodiert werden. Das DD-Statement stellt die Verbindung zwischen dem Programm und der Umgebung her.

Das Programm kennt nur den DD-Namen des DD-Statements. Die Parameter des DD-Statements legen fest, welchen Datenbestand vom Programm bearbeitet wird. Erst diese Einrichtung macht es möglich, einen Datenträger vom Programm zu entkoppeln.

Das folgende Beispiel zeigt, wie ein DD-Statement geschrieben wird:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//SYSUT1   DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=SHR
//SYSUT2   DD DSN=HLQ11.HLQ12.HLQ13,
//            DISP=(NEW,CATLG,DELETE),
//            DSORG=PS,
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb. 128: DD-Statements in einer Job Control

Normalerweise greifen Programme auf Datenbestände zu, welche auf einem externen Datenträger abgelegt sind. Es gibt allerdings die Möglichkeit, Datenbestände mit der Job Control zusammen dem Betriebssystem zur Verfügung zu stellen.

Diese Daten werden beim Einlesen durch die Readerfunktion von der eigentlichen Job Control getrennt und separat im Spoolbereich abgelegt. Die Abgrenzung von der Job Control erfolgt durch die positionellen Parameter. Bei Ausführung des Programms greift das Betriebssystem auf diese Datenbestände im SPOOL, genauso wie auf jeden anderen Datenbestand, zurück.

Das Ende der Daten im Eingabestrom wird

- durch das nächste JCL-Statement

oder

- mit der Zeichenkette „/*“ angezeigt.

Dieser Begrenzer (/*) wird auch als EOF (End of File) bezeichnet.

In der täglichen Praxis kommt es vor, dass einzelne Datenbestände aus einem Programm heraus nicht angesprochen werden sollen. Trotzdem müssen die entsprechenden DD Statements angegeben werden, denn das Programm macht einen OPEN auf die entsprechenden Files. Allerdings kann das Betriebssystem auch überlistet werden:

- Mit Hilfe des positionellen Parameters DUMMY wird dem Betriebssystem gesagt, dass mit „ddname“ kein externer Datenbestand verbunden werden soll.

In Abhängigkeit davon, ob es sich um einen Eingabe- oder einen Ausgabedatenbestand handelt, passieren unterschiedliche Dinge:

- Aufgrund der Beziehung zwischen DD-Namen und Datasetname wird die Verbindung von einem Puffer zu einem Datenbestand hergestellt.

- Bei Inputdatenbeständen wird durch den Parameter DUMMY sofort eine End-of-File-Marke in den Puffer hineingestellt. An dieser EOF-Marke erkennt das Programm, dass der Datenbestand zu Ende ist, wie das folgende Beispiel zeigt:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//SYSUT1   DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            DSORG=PS,
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb. 129: DUMMY im SYSIN-DD-Statement

- Normalerweise wird die in Abbildung 129 gesetzte EOF-Marke erst am Ende eines Datenbestandes durch das Betriebssystem in den Puffer eingestellt.

- Bei der Ausgabe werden zunächst durch die Ausgabebefehle einzelne Datensätze in den Puffer gestellt.

- Ist ein Puffer gefüllt, wird normalerweise der im Puffer befindliche Block auf den im DD-Statement spezifizierten Datenbestand übertragen.

- Durch den Parameter DUMMY wird diese Übertragung unterbunden. Man kann auch sagen, der Block wandert in den Mülleimer.

Das folgende Beispiel zeigt, wie der „*“ Parameter eingesetzt werden kann:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//SYSUT1   DD *
 Diese Zeilen sind Eingabedaten für das Programm
 IEBGENER. IEBGENER liest die Daten über den
 DD-Namen SYSUT1.
/*
//SYSUT2   DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            DSORG=PS,
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb. 130: *-Parameter im DD-Statement

Das Ende der Daten im Eingabestrom wird

- durch das nächste JCL-Statement

oder

- mit der Zeichenkette „/*“ angezeigt.

Eine Problematik tritt allerdings auf, wenn die Daten in Spalte eins und zwei jeweils einen Schrägstrich enthalten. Das erste Auftreten wäre gleichbedeutend mit dem Ende der Eingabedaten; und das ist nicht erwünscht.

Mit DD „*“ können deshalb keine Daten verarbeitet werden, die JCL Statements enthalten. Dazu dient der Parameter „DATA,DLM=%%“. Das Ende der Daten im Eingabestrom nach „DATA,DLM=%%“ wird jetzt ausschließlich durch den Delimiter „%%“ angezeigt. „%%“ ist hier als Platzhalter zu sehen, da der Programmierer den Delimiter selbst vergeben muss. Als „Delimiter“ sind nur Buchstaben, Ziffern und National Characters ($, #, @) erlaubt.

Das folgende Beispiel zeigt, wie mit dem Parameter „DATA,DLM=%%“ Daten verarbeitet werden, die JCL-Statements enthalten:

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//SYSUT1   DD *,DLM=$$
 Diese und die folgenden Zeilen sehen zwar aus
 wie JCL-Statements, stellen aber Daten dar, die
 vom Programm IEBGENER gelesen werden.
//Auch diese Zeilen sind Daten, trotz der
//Schrägstriche in den Spalten eins und zwei.
/*Auch diese Zeilen sind Daten, da ausschließ-
/*lich $$ als EOF interpretiert wird.
Das Programm liest die Daten über den DD-Namen
SYSUT1.
$$
//SYSUT2   DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            DSORG=PS,
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
```

Abb 131: Delimiter im DD-Statement

Das Ende der Daten im Eingabestrom wird jetzt ausschließlich durch die Zeichenkette '$$' in Spalte eins und zwei angezeigt. Die Vergabe der Delimiter-Zeichen ist vom Programmierer frei wählbar. Er muss nur darauf achten, dass die Delimiter-Zeichen am Anfang und am Ende des Datenstroms gleich sind.

Der Delimiter-Parameter kann auch in Kombination mit dem positionellen Parameter '*' verwendet werden, dann allerdings ohne die Zeichenketten '//' und '/*' in Spalte eins und zwei.

9.3 DDNAME – Der Dataset-Definitionsname

Der DD-Name kann aus maximal acht alphanumerischen Zeichen (A – Z, 0 – 9) oder Sonderzeichen (@ $ #) bestehen. Die erste Stelle darf keine Ziffer sein. Jeder DD-Name innerhalb eines Steps sollte unbedingt eindeutig sein. Tritt derselbe Name trotzdem in demselben Step mehrfach auf, so werden zwar alle Zuordnungen für jedes Statement gemacht, doch eventuelle Bezugsnahmen werden an das erste Statement des identischen Namens gerichtet. Die DD-Namen hängen vielfach vom verwendeten Compiler ab[23].

9.3.1 DSN – Der Dataset-Name

Der Parameter DSN gibt dem Dataset einen Namen. Dataset können temporär oder permanent existieren. Temporäre Datasets werden während eines Joblaufes erzeugt und verbleiben nur so lange im Speicher, wie der Job,

[23] Vgl. G.D.Brown: JCL Job Control Language im Betriebssystem MVS OS/390, 3. Aufl., S. 125

der das Dataset angelegt hat, aktiv ist. Wird der Job beendet, wird auch das temporär erzeugte Dataset wieder gelöscht.

Anders ist das bei einem permanenten Dataset. Hier wird ein Katalogeintrag erstellt und das Dataset auf einen physikalischen Datenträger geschrieben. Über den Katalogeintrag wird das Betriebssystem in die Lage versetzt, die Datei bei Bedarf wieder zu finden.

Ein Katalogeintrag erfüllt dabei die gleiche Aufgabe wie das Inhaltsverzeichnis eines Buches. Über das Inhaltsverzeichnis kann der Leser bestimmte Stellen aus einem Buch gezielt heraussuchen. Der Katalog eines Dateisystems hat Informationen darüber gespeichert, wo auf der Festplatte sich die einzlenen Dateisegmente befinden. Denn eine Datei wird in den seltensten Fällen zusammenhängend auf einen Datenträger geschrieben.

9.3.2 DISP – Die Dataset-Disposition

Der DISP-Parameter gibt an, wie auf ein Datenbestand zugegriffen werden soll. Bei bestehenden Dataset, deren Inhalt nur als Eingabedatenbestand benutzt wird, wird in der Regel die Disposition „SHR“. Diese Dispositon ermöglicht den gleichzeitigen Zugriff mehrerer Jobs auf eine Datei. Bei den weiteren Dispositionsangaben ist die Frage zu klären, was mit dem Datenbestand passieren soll. Die folgende Tabelle zeigt eine Auflistung der möglichen Dispositionsangaben und erläutert diese.

Disposition	**Bedeutung**
SHR	Nur Lesezugriff
OLD	Exklusiver Lese-/Schreibzugriff; beim Schreiben wird der ursprüngliche Inhalt überschrieben
MOD	Exklusiver Lese-/Schreibzugriff; beim Schreiben wird der ursprüngliche Inhatl nicht überschrieben. Der neue Inhalt wird an das Dateiende geschrieben.
(NEW,CATLG,DELETE)	Datei wird neu angelegt und beim normalen Stepende katalogisiert. Bricht der Step ab, wird die Datei wieder gelöscht.
(OLD,DELETE,KEEP)	Datei wird mit exklusivem Lese-/Schreibzugriff angesprochen und gelöscht. Bricht der Step ab, wird die Datei aufgehoben.
(NEW,PASS)	Datei wird neu angelegt und weitergereicht. Diese Angabe wird bei temporären Dataset gemacht.

Tab. 15: Datei-Dispositionen

9.3.3 DSORG – Dataset-Organisation

Eine DSORG-Angabe wird mitgegeben, um die Art der Dateiorganisation zu definieren. Standardannahme ist DSORG=PS. Die meisten Anwendungsprogramme geben die Angabe für DSORG mit, so dass dieser Parameter in der JCL fast nie angegeben werden muss. Die gängigsten Organisationsarten lauten:

- PS = Sequentiell (Physical Sequential)
- PO = Partitioned (Partitioned Organized)

9.3.4 RECFM – Das Record-Format

Das Record-Format bezieht sich auf die Formatierung eines Datenbestandes. Es werden hauptsächlich zwei Formatierungen unterschieden:

- FB (Fixed Blocked)
- VB (Variabel Blocked)

Die Blockung ist für die Länge eines Datensatzes interessant. Bei einer Datei mit fixer Blockung und einer definierten Satzlänge (LRECL) von 80 Zeichen werden bei einem Datensatz, der nur 40 Zeichen hat, die Stellen 41 bis 80 mit Blanks ergänzt, so dass auch dieser Datensatz eine Satzlänge von 80 Zeichen erreicht.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 132: RECFM mit fixer Blockung

Bei einer variablen Blockung ist ein Datensatz, der 40 Zeichen hat, auch nach 40 Zeichen zu Ende. Es werden keine „Füllaktionen“ gestartet. Außerdem beginnt ein Datensatz in einer variabel geblockten Datei erst ab Stelle fünf, da die ersten vier Stellen Informationen zum eigentlichen Datensatz enthalten, das sogenannte Record-Descripter-Word.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=VB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 133: RECFM mit variabler Blockung

9.3.5 LRECL – Die logische Record-Länge

Die logische Record-Länge wird bei der Datei-Allokation definiert und legt die maximale Länge eines Datensatzes fest. Die Attribute RECFM und LRECL

stehen dabei im Verhältnis zueinander. Wie in Kapitel 8.3.4 erläutert entscheidet das Record-Format darüber, ob ein zu kurzer Datensatz mit Blanks ergänzt wird oder nicht.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 134: LRECL mit fixer Blockung

Das Beispiel in Abbildung 134 zeigt eine Job Control, bei der eine Datei mit einer logischen Satzlänge von 80 Zeichen und fixer Blockung erzeugt wird. Datensätze mit kürzerer logischer Satzlänge werden mit Blanks ergänzt, so dass auch diese am Ende die logische Satzlänge von 80 Zeichen erreichen.

Abbildung 135 zeigt eine Job Control, bei der eine Datei mit variabler Blockung und einer logischen Satzlänge von ebenfalls 80 Zeichen definiert wird.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            RECFM=VB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 135: LRECL mit variabler Blockung

In diesem Beispiel werden Datensätze mit weniger als 80 Zeichen nicht mit Blanks aufgefüllt. Der Datensatz endet nach dem letzten Zeichen, d.h. hat ein Datensatz nur 15 Zeichen ist dieser auch nach dem 15. Zeichen zu Ende. Desweiteren beginnt ein Datensatz nicht auf Stelle eins wie in der fixed geblockten Datei, sondern auf Stelle fünf. Die Stellen eins bis vier sind für das Record-Descripter-Word vorgesehen.

9.3.6 DCB – Der Data Control Block-Parameter

Unter dem DCB-Parameter werden in der Regel RECFM und LRECL zusammen gefasst angegeben. Ein weiterer Parameter, der in Verbindung mit dem DCB-Parameter angegeben wird ist die Blocksize (BLKSIZE). Da diese Angabe heutzutage nicht mehr so häufig eingesetzt wird, soll hier auch nicht näher darauf eingegangen werden.

Die folgende Abbildung zeigt eine Job Control, bei der die gleichen Angaben wie in Abbildung 136 gemacht wurden, nur dass hier die Angabe des Record-Formates und der Record-Länge unter einer DCB-Angabe zusammen gefasst worden sind.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEFBR14
//DD01     DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(RECFM=FB,LRECL=80),
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 136: DCB-Parameter

9.3.7 Rückbezug

Mit Hilfe des Rückbezugsparameter kann man sich auf Datei-Attribute beziehen, ohne diese wiederholt angeben zu müssen. Vorteilhaft ist dies dann, wenn eine Datei kopiert wird und der Programmierer sich auf die Attribute der Eingabedatei beziehen kann. Auf diese Weise erspart sich der Programmierer die Arbeit, die Attribute der Ausgabedatei, welche im Normalfall äquivalent der der Eingabedatei sind, erneut definieren zu müssen.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//SYSUT1   DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=SHR
//SYSUT1   DD DSN=HLQ11.HLQ12.HLQ13,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=*.SYSUT1,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 137: Rückbezugsparameter

Im Beispiel der Abbildung 137 erspart man sich durch die DCB-Angabe den Aufwand, „LRECL=*.SYSUT1“ und „RECFM=*.SYSUT1“ einzeln eingeben zu müssen. Das Betriebssystem erkennt durch die DCB-Angabe automatisch dass es sich um das Record-Format und die logische Satzlänge handelt und holt sich die Attribut-Informationen aus der Eingabedatei im DD-Statement SYSUT1.

Das folgende Beispiel zeigt eine Job Control, die eine Datei löscht und wieder anlegt. Dieses Beispiel soll weitere Möglichkeiten aufzeigen, wie der Rückbezugsparameter eingesetzt werden kann, ist jedoch für die Praxis nicht empfehlenswert.

```
//JOB00001 JOB (123456),SCHEIBE,
//             CLASS=A,
//             MSGCLASS=T,
//             NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEFBR14
//DELETE   DD DSN=HLQ11.HLQ12.HLQ13,
//            DISP=(OLD,DELETE,DELETE),
//            RECFM=FB,
//            LRECL=80,
//            SPACE=(80,(1,10000),RLSE)
//*
//STEP2    EXEC PGM=IEBGENER
//SYSUT1   DD DSN=HLQ1.HLQ2.HLQ3,
//            DISP=SHR
//SYSUT1   DD DSN=HLQ11.HLQ12.HLQ13,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=*.STEP1.DELETE,
//            SPACE=(80,(1,10000),RLSE)
```

Abb. 138: Rückbezugsparameter als Negativbeispiel

Warum ist das Beispiel in Abbildung 138 nicht empfehlenswert?

Sobald der Job gestartet wird ist es zwingend erforderlich, dass die Datei „HLQ11.HLQ12.HLQ13“, die im STEP1 gelöscht werden soll, vorhanden ist, eben damit sie gelöscht werden kann. Ist diese Datei nicht vorhanden wird der Job mit „JCLERR“ abbrechen und die Meldung „DATA SET NOT FOUND“ ausgeben. Alle Nachläuferjobs würden konsequenterweise auch nicht anlaufen.

Weiterhin könnte diese Art JCL zu erstellen sowohl für neue Mitarbeiter in der Einarbeitungsphase als auch für Anfänger sehr schwer zu verstehen sein und dadurch Nährboden für Fehler bieten. Generell sollte ein Programmierer überlegen, ob er eine JCL so generiert, dass sie durch Rückbezugsparameter auf bereits vorhandenene Parameter zurückgreift oder ob er der Einfachheit halber nicht redundante Parameterangaben bevorzugen sollte um einen Job transparent zu halten.

10 System Output Dataset

Neben den positionellen Parametern gibt es auch noch eine Vielzahl von Schlüsselwortparametern mit den unterschiedlichsten Aufgaben.

Prinzipiell kann man die Schlüsselwortparameter in drei große Gruppen einteilen:

- Parameter zum Spool-Output
- Parameter zum Anlegen und Zuordnen von Datenbeständen
- Referenzen auf andere DD-Statements

10.1 Spool-Output

Normalerweise werden Ausgabedatenbestände auf einen externen Datenträger geschrieben. Es gibt aber auch die Möglichkeit, die Ausgaben auszudrucken. Dies kann nicht direkter geschehen, da der Drucker ein relativ langsames Ausgabegerät ist.

Aus diesem Grund werden alle Druckausgaben eines Jobs zunächst einmal im Spoolbereich gesammelt. In der Outputphase des JES werden die Ausgaben für den gesamten Job dann ausgedruckt. Dies geschieht zusammen mit den Protokollen des Jobs. Aus diesem Grund nennt man diesen Datenbestand im Spool auch das

- System Output Dataset.

Der Parameter, mit welchem dem Betriebssystem mitgeteilt wird, dass der Datenbestand ein Spoolausgabe-Datenbestand ist, lautet

- SYSOUT.

Normalerweise erfolgt die Ausgabe des Spool-Output nach Ende der gesamten Jobverarbeitung durch das JES in der Outputphase. Alle zum Druck anstehenden Listen werden ausgegeben. Die Druckausgabe lässt sich mit den folgenden Parametern regeln:

```
HOLD={YES}
     {Y}
     {NO}
     {N}
```

Abb. 139: Druckausgabe-Regelung über den HOLD-Parameter

Durch den Parameter HOLD kann man dem JES explizit für einen Datenbestand mitteilen,

- die Standardeinstellung für diesen Parameter ist NO.
- HOLD=YES verhindert den Druck des SYSOUT-Dataset, bis er vom Operator freigegeben wird.

Dieser Parameter ist vor allem für den Test sehr nützlich. Der TSO-User kann die Liste erst mal am Bildschirm begutachten, bevor sie gedruckt wird.

```
FREE={END}
     {CLOSE}
```

Abb. 140: Druckausgabe-Regelung über den FREE-Parameter

Wie durch den Parameter HOLD die Ausgabe einer Liste verzögert werden kann, kann der Druck einer einzelnen Liste durch den Parameter FREE beschleunigt werden. Die Standardeinstellung für diesen Parameter ist END, d.h. die Ausgabe erfolgt nach Ende des Jobs.

FREE=CLOSE besagt, dass dieser Datenbestand ausgedruckt werden kann, sobald durch das Programm der CLOSE auf den Datenbestand erfolgt ist. Datenbestände, die über SYSOUT ausgegeben werden, werden normalerweise nur ein einziges Mal erzeugt. Ist eine Liste mehrfach gewünscht, kann auch der Parameter COPIES verwendet werden.

```
COPIES={nnn}
```

Abb. 141: Druckausgabe-Regelung über den COPIES-Parameter

Mit dem Parameter COPIES kann festgelegt werden, wie oft das Sysout Dataset gedruckt werden soll.

Eine weitere, wichtige Aufgabe des DD-Statements ist das Anlegen und Zuordnen von Datenbeständen.

```
//RAUS     DD SYSOUT=A,
//             HOLD=YES
```

Abb. 142: Druckausgabe SYSOUT=A

Durch die Angabe SYSOUT=A in Abbildung XX werden die Daten, die auf das File RAUS ausgegeben werden, über den Spoolbereich auf einen Drucker geleitet, der für die Ausgabeklasse A eröffnet ist.

Die Klasse kann irgendein alphanumerisches Zeichen (A bis Z, 0 bis 9) oder der Stern '*' sein. In der Installation werden üblicherweise Klassifizierungen der Ausgabe vorgenommen:

- Ausgabe in sehr großer Menge
- Ausgabe mit hoher Priorität
- Ausgabe auf besonderem Formular
- usw.

Der Stern '*' gibt an, daß dieselbe Klasse zu verwenden ist, die im JOB-Statement als Parameter MSGCLASS angegeben ist[24], wie das folgende Beispiel verdeutlicht:

[24] Vgl. G.D.BROWN: JCL Job Control Language im Betriebssystem OS/390 MVS, 3. Aufl., S. 175 ff

```
//JOB00001 JOB (123456),SCHEIBE,CLASS=A,
//             MSGCLASS=T,NOTIFY=&SYSUID
…
//SYSPRINT DD SYSOUT=*
```

Abb. 143: Druckausgabe mit Informationen aus der MSGCLASS

Die in Abbildung 143 gemachte Angabe „SYSPRINT DD SYSOUT=*“ entspricht also der Druckausgabe „SYSPRINT DD SYSOUT=T“.

```
//RAUS     DD SYSOUT=A,
//             HOLD=YES
```

Abb. 144: Druckausgabe HOLD=YES

Durch die Angabe HOLD=YES in Abbildung XX wird der Druck des SYSOUT Datasets verhindert, bis er vom Operator freigegeben wird.

```
//RAUS     DD SYSOUT=A,
//             FREE=CLOSE
```

Abb. 145: Druckausgabe FREE=CLOSE

FREE=CLOSE in Abbildung 145 besagt, dass dieser Datenbestand ausgedruckt werden kann, sobald durch das Programm der CLOSE auf den Datenbestand erfolgt ist.

```
//DRUCK    DD SYSOUT=A,
//             COPIES=5
```

Abb. 146: Druckausgabe COPIES=5

Die Liste, die unter dem DD-Namen DRUCK in Abbildung 146 ausgegeben wird, wird in fünffacher Ausfertigung erzeugt. Der COPIES-Parameter ist allerdings nur aktiv bei direkter Ausgabe. Wenn die Ausgabe in eine Held Output Queue geschrieben wird, wird der Parameter COPIES ignoriert.

10.2 JESMSGLG

Nach jedem Joblauf ist es wichtig zu erfahren, wie sich ein Job während seiner aktiven Phase auf der Maschine verhalten hat. Es werden Informationen benötigt, die darüber Aufschluss geben, ob ein Job korrekt durchgelaufen ist oder ob es Probleme gab. Diese Aufgabe, derartige Informationen aufzunehmen, übernimmt das JESMSGLG.

Im JESMSGLG wird das vom System dokumentierte „Jobverhalten“ unter der Überschrift „Job-Log“ gespeichert. Hier kann nachgelesen werden, welche RCs die einzelnen Steps nach ihrer Ausführung ausgegeben haben bzw. ob diese überhaupt ausgeführt worden sind.

Desweiteren kann über das JESMSGLG der Name des Programmierers eingesehen werden, der die JCL erstellt hat. In der Praxis ist diese Angabe

speziell für das Personal im Operating wichtig, da diese gezielt reagieren und die entsprechende Person und über den Abbruch informieren können.

Das System erstellt eine Job-Statistik, welche

→ die Anzahl der gelesenen Karten,
→ die Anzahl der Druckdatensätze
→ die Anzahl der eingelesenen Datensätze und
→ die Laufzeit eines Jobs

dokumentiert und gibt diese über das JESMSGLG aus. Hier kann jeder, der über ein Programm wie SDSF, IOF o.ä. verfügt und Zugriffsrechte auf beendete hat, diese Informationen einsehen. Ein Beispiel für ein JESMSGLG-Output befindet sich auf der folgenden Seite.

Abb. 147: Normales Job-Log

```
******************************** TOP OF DATA ********************************************************************
 D E B A U B G M
                J E S 2  J O B  L O G  --  S Y S T E M  B B G P  --  N O D E

10.22.31 JOBXXXXX ---- FRIDAY,    16 FEB 2007 ----
10.22.31 JOBXXXXX  IRR010I  USERID USER01   IS ASSIGNED TO THIS JOB.
10.22.33 JOBXXXXX  ICH70001I USER01   LAST ACCESS AT 10:21:38 ON FRIDAY, FEBRUARY 16, 2007
10.22.33 JOBXXXXX  $HASP373 JOB00001 STARTED - INIT TTS2 - CLASS A - SYS SYSTEM
10.22.33 JOBXXXXX  IEF403I JOB00001 - STARTED - TIME=10.22.33
10.22.33 JOBXXXXX  -                                          --TIMINGS (MINS.)--            ----PAGING COUNTS---
10.22.33 JOBXXXXX  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB CLOCK   SERV  PG  PAGE  SWAP   VIO SWAPS
10.22.33 JOBXXXXX  -JOB00001 STEP1               00     20      4    .00    .00    .0    422   0     0     0     0     0
10.22.33 JOBXXXXX  -JOB00001 STEP2               00      0      0    .00    .00    .0      0   0     0     0     0     0
10.22.33 JOBXXXXX  IEF453I JOB00001 – ENDED - TIME=10.22.33
10.22.33 JOBXXXXX  -JOB00001 ENDED.  NAME-SCHEIBE M           TOTAL TCB CPU TIME=   .00  TOTAL ELAPSED TIME=    .0
10.22.33 JOBXXXXX  $HASP395 JOB00001 ENDED
------ JES2 JOB STATISTICS ------
  16 FEB 2007 JOB EXECUTION DATE
           24 CARDS READ
           89 SYSOUT PRINT RECORDS
            0 SYSOUT PUNCH RECORDS
            5 SYSOUT SPOOL KBYTES
         0.00 MINUTES EXECUTION TIME
******************************** BOTTOM OF DATA *****************************************************************
```

10.3 JESYSMSG

Das JESYSMSG ist die detailliertere Darstellung zum JESMSGLG. Hier wird aufgeschlüsselt, welche Datei bereits vorhanden war und dadurch der Job mit JCLERR abgebrochen ist. Desweiteren wird der Operator bei Space-Problemen über die Angabe in der Reihenfolge

➔ Jobname, Stepname, DD-Statement, Dateiname

explizit an die Datei herangeführt, die mit zu geringem Space definiert wurde und folglich der Job mit B37 abgebrochen ist.

Für die typischen JCL-Fehler wie JCLERR, B37 usw. ist dieses Log sehr nützlich, da es die Zeile und den genauen Fehler benennt. Für Fehler, die aufgrund von Programmen aus der eigenen Anwendungsentwicklung aufgetreten sind ist es allerdings wenig hilfreich.

11 Wie werden Daten tatsächlich auf Platte abgelegt?

Eine Festplatte ist ein magnetooptisches Speichermedium und gehört zu den externen Datenträgern.Sie besteht aus mehreren Scheiben. Auf diese Scheiben werden die Daten in konzentrischen Kreisen abgelegt. Jeder dieser Kreise wird als eine Spur bezeichnet. Um die Festplatte vor äußeren Umwelteinflüssen zu schützen ist sie in einem Gehäuse untergebracht.

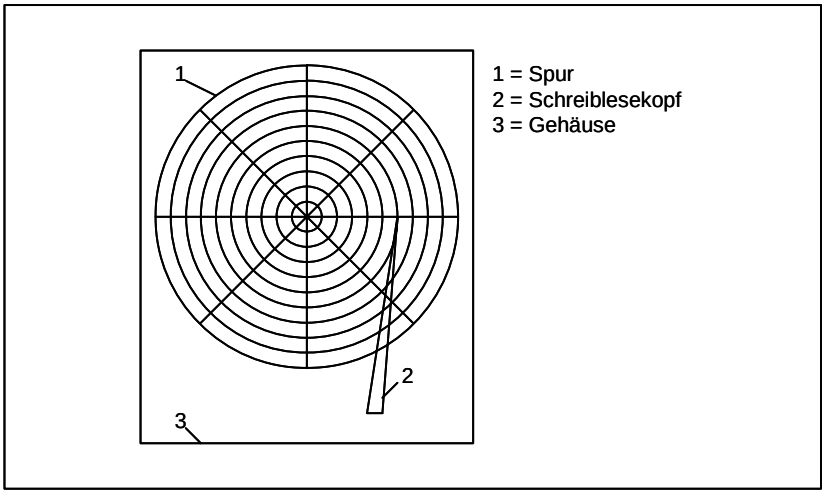

Abb. 148: Logische Skizzierung einer Festplatte

Im Gehäuse integriert ist ein Schreiblesekopf, welcher, auf einem Luftkissen schwebend, die Festplatte nach magnetischen und nicht magnetischen Feldern abtastet. Auf diese Weise können Bitfolgen von '0' und '1' zusammenhängend ermittelt und ausgewertet werden. Eine Komplettfolge von acht Bit ergeben ein Byte und somit einen alphabetischen oder numerischen Wert.

In Abhängigkeit vom Datenträgertyp beinhalten die Scheiben eine unterschiedliche Anzahl von Spuren. Bei modernen Datenträgertypen sind mehrere Scheiben auf einer Achse zusammengefasst. Die Spuren, die über alle Scheiben hinweg untereinander liegen, werden unter dem Begriff Zylinder zusammengefasst. Auch hier ist die Anzahl der Spuren, die zu einem Zylinder gehören, abhängig vom Datenträgertyp.

Die folgenden Angaben sind vom jeweiligen Datenträgertyp abhängig:

- Anzahl Byte pro Spur (= Spurkapazität)
- Anzahl der Spuren je Zylinder
- Anzahl der Zylinder

Da die gesamte Verwaltung und Organisation einer Platte auf Spurebene passiert, ist die Spur die kleinste adressierbare Einheit auf der Platte.

12 Nützliche Utilities

Utilities (engl. = Nutzen) sind Dienst- oder Hilfsprogramme, die vom Hersteller des Betriebssystems mitgeliefert werden, um wiederkehrende Aufgaben zu erfüllen. Eine dieser Aufgaben ist beispielsweise das Kopieren von Datenbeständen.

Eine andere, häufige Aufgabe ist die Reorganisation von Datenbeständen. Darunter versteht man die Freigabe von nicht mehr genutztem Platz auf Datenspeichern.

Weitere Aufgaben sind u.a.:

- das Verändern des Aufbaus von Datensätzen, z.B. um ungenutzten Platz freizugeben oder um einen Datenbestand an neue Erfordernisse anzupassen.
- das Vergleichen von kopierten Datenbeständen, z.B. um fehlerhafte Kopien zu erkennen.
- die Anlistung von Datei-Ausschnitten, z.B. um die erste Zeile jedes Members einer untergliederten Datei lesen zu können.
- die Anlistung des Datei-Aufbaus, z.B. um Reorganisationsmaßnahmen zu planen.
- die Anlistung des Aufbaus einer Magnetplatte, z.B. um die Menge freien Speicherplatzes festzustellen.

Die folgende Tabelle enthält die IBM-Standard-Utilities:

Aufgaben	Programm
Kopieren von Datenbeständen	IEBCOPY, IEBGENER, ICEMAN, ICEGENER
Reorganisation	IEBCOPY
Vergleichen von Datenbeständen	IEBCOMPR
Verändern des Datensatzaufbaus	IEBGENER
Anlistung von Dateiinhalten	IEBPTPCH
Veränderung von Dateiinhalten	IEBUPDTE
Anlistung des Datei-Aufbaus	IEHLIST
Sortieren von Dateiinhalten	ICEMAN

Tab. 16: IBM-Standard-Utilities

Eine Sonderstellung nimmt das Programm 'IEFBR14' ein, das zwar häufig eingesetzt wird, aber eigentlich nicht als Utility bezeichnet werden kann. Das folgende Beispiel zeigt, wie ein Utility aufgerufen wird:

```
//JOB00001 JOB …
//*
//STEP1    EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
//…
```

Abb. 149: Aufruf eines Utilities

Das Utility 'IEBGENER' wird über seinen eigenen Namen aufgerufen. Man gibt also im EXEC-Statement als Programmnamen den Namen des Utilities an. Bei allen Utilities sind die DD-Statements mit den DD-Namen SYSIN und SYSPRINT erforderlich, wobei zu bemerken ist, dass die DD-Namen unter Umständen von Utilitiy zu Utility abweichen können. Das Prinzip bleibt jedoch unverändert. Diese Angaben haben beim Aufruf von Utilities eine besondere Bedeutung:

- SYSPRINT wird für die Ausgabe der Verarbeitungsprotokolle verwendet. Die Ausgabe besteht aus 133-stelligen Sätzen mit (ASA-) Vorschubsteuer-zeichen in Spalte 1.
- SYSIN ist für die Eingabe der Kontrollwörter und Parameter notwendig. Diese Eingabe erfolgt entweder innerhalb eines Job-Strams oder über eine beliebige Datei mit der Satzlänge 80.

Es ist außerdem möglich, diese DD-Statements auf DUMMY zu setzen, wenn z.B. keine Eingabe gemacht oder die Standardverarbeitung gewünscht wird oder die Protokolle nicht interessieren.

Bei jedem Kontrollwort, das unter SYSIN eingegeben wird, ist es zulässig, ein LABEL anzugeben, über das dieser Befehl identifiziert wird. Damit ist es möglich, bei einer größeren Anzahl von angeforderten Funktionen diese namentlich zu unterscheiden.

Eine Abfrage oder Verzweigung mit Hilfe dieser LABEL ist allerdings nicht möglich und damit können sie nur zur Identifizierung benutzt werden. Ob sie geschrieben werden oder nicht, bleibt dem Programmierer überlassen.

```
//JOB00001 JOB …
//*
//STEP1    EXEC PGM=Utility
//SYSPRINT DD SYSOUT=*
//name1    DD DSN=Dateiname,
//            DISP=SHR
//name2    DD DSN=Dateiname,
//            DISP=SHR
//SYSIN    DD *
//…
```

Abb. 150: Angabe von DD-Namen

Mit den DD-Namen 'name1' und 'name2' werden die zu verarbeitenden Datenbestände dem Utility bekanntgegeben. Bei einigen Utilities können diese Namen beliebig vom Programmierer vergeben werden, bei anderen sind sie vorgegeben und müssen eingehalten werden.

13 Das Protokoll

Das JES stellt bei der Ausführung eines Jobs einen Bericht zusammen. Dieser zeichnet alle Aktivitäten auf, die vom JES oder dem Betriebssystem für den Job geleistet werden. Dieser Bericht wird im Spoolbereich aufgebaut.

Mit dem Parameter MSGCLASS kann im voraus bestimmt werden, ob das SYSOUT-Dataset nach Jobende gedruckt oder im Spoolbereich gehalten wird, bis es gelöscht oder zur Ausgabe freigegeben wird.

Der Bericht ist in drei Blöcke gegliedert:

- JOB LOG gibt einen groben Überblick zum Lauf des Jobs
- JCL zeigt die Auflösung der JCL-Statements
- MESSAGES enthält detaillierte Informationen zum Einsatz