
Oldenbourg

Herausgegeben von
Dr. Klaus Becker-Berke

MVS/ESA JCL

Einführung in die Praxis

von
Michael Winter
Deutsche Lufthansa AG

3., korrigierte und aktualisierte Auflage

Oldenbourg Verlag München Wien

Michael Winter sammelte nach seinem Studium mehrjährige Erfahrung als Anwendungsentwickler in der IBM-Welt in den Umgebungen DOS/VSE und MVS/TSO, bevor er als EDV-Dozent zur Deutschen Lufthansa AG wechselte. Er ist dort verantwortlich für die Schulung der Programmierer in den Bereichen TSO/ISPF, MVS, JCL und REXX sowie systemnaher Software.

Anschrift:
Michael Winter
Lufthansa Systems GmbH
Am Weiher 24
65451 Kelsterbach
Tel.: +49 69 696 6587
Fax: +49 69 696 6588
E-Mail: Michael.Winter@dhl.de

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Winter, Michael:
MVS ESA JCL : Einführung in die Praxis / von Michael Winter.
[Hrsg. von Klaus Becker-Berke.]. – 3., korrigierte und aktualisierte
Aufl. – München ; Wien : Oldenbourg, 1999
ISBN 3-486-25058-2

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0, Internet: http://www.oldenbourg.de

Lektorat: Birgit Zoglmeier
Herstellung: Rainer Hartl
Umschlagkonzeption: Kraxenberger Kommunikationshaus, München
Gedruckt auf säure- und chlorfreiem Papier
Gesamtherstellung: R. Oldenbourg Graphische Betriebe GmbH, München

Inhaltsverzeichnis

Vorwort

Von einem, der auszog,
JCL zu lehren

Mal ganz ehrlich: Wie sind Sie zu Ihrer ersten JCL gekommen?

War es nicht auch so, daß ein netter Kollege Ihnen anbot, seine JCL zu kopieren. "Da ändern Sie mal meine Userid in Ihre und dann wird das Ganze schon funktionieren".
Und es funktionierte. Manchmal. Manchmal funktionierte es auch nicht und der nette Kollege hatte gerade Urlaub. Dann war man gezwungen, Änderungen an Dingen vorzunehmen, die man womöglich nicht verstand. Die Resultate sind meist dementsprechend.
Erstaunlicherweise gibt es heutzutage noch viele Verantwortliche in der DV-Industrie, die der Meinung sind, man könne JCL so eben nebenbei lernen. Gerade für Anwendungsprogrammierer wird eine JCL-Ausbildung oft für überflüssig gehalten.
Die Folgen dieser Haltung konnte ich am eigenen Leib erfahren. Auch ich bin in der oben beschriebenen Weise zum ersten Mal mit JCL in Berührung gekommen. Eine gezielte JCL-Ausbildung gab es nicht. 'Learning by doing' war angesagt, meistens war es jedoch 'Learning by surprise' oder 'Learning by ABEND'. Ich habe so ziemlich jeden JCL-Fehler und jeden ABEND mitgemacht. Doch diese Art, JCL zu lernen, ist sehr langwierig und ineffizient. Damals hätte ich ein Buch, wie es jetzt hier vorliegt, gut gebrauchen können.
Dieses Buch ist aus den Erfahrungen der Praxis entstanden. Es vermittelt Ihnen die Kenntnisse, die Sie brauchen, um mit JCL sicher umgehen zu können. Es vermittelt auch wichtige Zusammenhänge. Sie sollen nicht nur Syntax beherrschen lernen, sondern Sie sollen vor allen Dingen verstehen, warum bestimmte Dinge so funktionieren, wie sie funktionieren.
Das Buch ist in zweierlei Hinsicht strukturiert. Zum einen im Hinblick auf die Parameter, die Sie einsetzen müssen, zum anderen im Hinblick auf bestimmte Funktionen und Aufgabenstellungen, die Ihnen in Ihrer täglichen Arbeit begegnen können.
Die Neuerungen, die sich mit der Version 4.2 (oder höher) von MVS/ESA ergeben haben, sind in diesem Buch berücksichtigt. Gleiches gilt für SMS, das Storage Management Subsystem, das in den letzten Jahren zu einem integralen Bestandteil der Datenverwaltung in der MVS-Welt wurde. Alle Neuerungen, die in Zusammenhang mit SMS stehen, werden vorgestellt.
Großen Wert habe ich auf die Vollständigkeit der Beispiele gelegt. So ist nicht nur die JCL erläutert, es wird auch gezeigt, wie das entsprechende Programmcoding auszusehen hat. Ich habe mich entschlossen, dies immer an COBOL-Beispielen darzustellen, da diese Programmiersprache in der IBM-Welt noch immer große Bedeutung hat.
Die im Buch verwendeten Begriffe und Abkürzungen werden bei ihrem ersten Auftauchen erläutert. Im Anhang befindet sich ein Glossar, in dem die wichtigsten Begriffe zusammengefaßt sind.
Sollten Sie Dinge in diesem Buch entdecken, die bei Ihnen so nicht funktionieren, so halten Sie Rücksprache mit Ihrer Systemprogrammierung. Es ist durchaus denkbar, daß bei

Ihnen Besonderheiten installiert worden sind. Ansonsten würde ich mich freuen, wenn Sie sich an den Verlag wenden würden. Dies gilt auch, wenn Sie bestimmte Dinge in diesem Buch erwartet und dennoch nicht gefunden haben. Ihre Anregungen können dann bei einer Neuauflage Berücksichtigung finden.
Ich wünsche, daß Ihnen dieses Buch einige Aha-Erlebnisse vermitteln wird. Wenn es Ihnen gelingt, JCL damit zu meistern, so würde mich dies am meisten freuen. Dann hätte sich die Arbeit des Lehrenden gelohnt.

Frankfurt, im März 1996

Michael Winter

Vorwort zur 3. Auflage

Sechs Jahre sind nunmehr seit dem Erscheinen der ersten Auflage dieses Buches vergangen. In dieser Zeit hat sich die EDV-Welt gravierend verändert. So stark sind Veränderungen, daß selbst die Prognosen der Fachleute über den Haufen geworfen wurden.

Es gibt Sie immer noch, entgegen aller Prognosen: Die IBM-Hosts. Auch sie haben sich angepaßt (wenngleich man das von der JCL nicht sagen kann, sie ist nun mal nicht so dynamisch). Sie werden die Jahrtausendwende erleben und, so hoffen alle, gut bewältigen.

Da es die Rechner immer noch gibt, müssen die Programme weiter laufen. Der Bedarf nach JCL ist also immer noch vorhanden. Ich hoffe, daß es Ihnen mit Hilfe dieses Buches gelingen wird, praktische Probleme mit Hilfe von JCL zu bewältigen.

Frankfurt, im März 1999

Michael Winter

1 Einleitung

Das vorliegende Buch befaßt sich mit Job Control Language - JCL. Um die Eigenheiten dieser Kontrollsprache zu verstehen, muß man wissen, daß man es bei JCL mit einem Produkt aus der Computer-Urzeit zu tun hat. JCL hat seine Ursprünge in den sechziger Jahren, als IBM die Computerserie der /360er Reihe einführte. Die JCL, die Sie heute benutzen, unterscheidet sich im Prinzip nicht von der, die es schon vor fast 30 Jahren gegeben hat. Dies ist auch einer der Gründe dafür, daß der Umgang mit JCL bisweilen schwerfällt.

JCL brauchen Sie, wann immer Sie **Batch**-Verarbeitung betreiben. Der Begriff Batch-Verarbeitung ist ebenfalls ein Relikt aus der EDV-Urzeit, wurde damit doch der physische Lochkartenstapel bezeichnet, mit dessen Hilfe man früher Jobs in das System einlas. Heute werden die JCL-Anweisungen selbstverständlich auf elektronischem Wege erzeugt. Der Begriff Batch-Verarbeitung ist jedoch geblieben.

Warum ist JCL so wichtig?

Wenn Sie auf einem IBM Großrechnersystem arbeiten, kommen Sie zwangsläufig mit JCL in Berührung. JCL wird gebraucht, um Anwendungsprogramme zu kompilieren und zu linken. Ohne JCL gibt es keine ausführbaren Programme. Anwendungsprogramme enthalten ausschließlich logische Datenbeschreibungen. Die Verbindung zu den physischen Daten erfolgt über JCL. Wenn Sie Funktionen, wie Kopieren von Datasets, nicht ständig manuell mit ISPF ausführen wollen, so brauchen Sie auch dazu JCL.

Im Gegensatz zu Ihrem Programm beschreibt JCL physische Daten. Zwar gibt es heute einen Trend, den Programmierer von den Zwängen der physischen Datensicht zu befreien, doch Sie sollten sich klarmachen, daß, auch wenn es wünschenswert ist, auf die physische Datensicht nicht verzichtet werden kann.

Eine optimale physische Datenstruktur entscheidet letztlich über die Performance eines Programms. Sie entscheidet auch über eine optimale Nutzung der Speicherressourcen.

Dagegen ist die Frage, ob strukturiert oder unstrukturiert programmiert wurde, für die Performance und die Speicherplatznutzung ohne jede Bedeutung.

Wie ist dieses Buch aufgebaut?

In diesem einleitenden Kapitel wird auf einige wichtige Grundlagen eingegangen, die zum Verständnis der Bedeutung der JCL-Parameter von Bedeutung ist.

Im zweiten Kapitel werden Ihnen alle wichtigen JOB-, EXEC- und DD-Anweisungen vorgestellt. Sie lernen, existierende Datasets anzusprechen und neue Datasets im Job zu erzeugen. Die unterschiedliche Syntax für das Anlegen auf Platte bzw. Band/Kassette wird erläutert. Sie erlernen auch den Umgang mit temporären Datasets und wie Sie diese Datasets an die folgenden Steps übergeben können.

Zur Vereinfachung wiederkehrender Abläufe lassen sich JCL-Prozeduren einrichten. Im Kapitel 3 lernen Sie, wie man vorhandene Prozeduren einsetzen kann und wie man eigene Prozeduren schreiben kann. Prozeduren geben Ihnen die Möglichkeit, Variable in der JCL zu definieren. Damit kann JCL sehr flexibel eingesetzt werden.

Für eine ganze Reihe von wichtigen Funktionen, gibt es sogenannte Utility-Programme. Sie werden benötigt, um z.B. mit dem Katalog richtig umgehen zu können oder Datasets zu kopieren. Auf alle wichtigen Utility-Funktionen wird im Rahmen des Kapitels 4 eingegangen.

Der Umgang mit VSAM-Datasets erfordert eine umfangreiche Syntax, auf die im Kapitel 5 eingegangen wird. Sie lernen, wie die verschiedenen Arten von VSAM Datasets angelegt werden. Außerdem wird die Möglichkeit, Alternativindices zu definieren, erläutert. Dabei werden auch Performanceaspekte berücksichtigt.

Wenn oben gesagt wurde, daß JCL in den letzten Jahren (Jahrzehnten) sehr statisch war, so gilt das nicht für SMS. Mit der Einführung vom SMS (System Managed Storage) ergeben sich eine Reihe von Änderungen für die Benutzer von JCL. Zudem werden mit SMS 10 neue JCL-Parameter eingeführt. Im Kapitel 6 wird deren Benutzung ebenso erläutert wie auch Probleme, die sich beim Umstieg von einer Nicht-SMS auf eine SMS-Umgebung ergeben können.

Im siebten Kapitel wird auf wichtige JCL-Schnittstellen eingegangen. Es wird gezeigt, wie Jobs auf andere Weise als durch einen SUBMIT aus einem Dataset heraus gestartet werden können. Dabei handelt es sich um die Möglichkeit, CLISTen oder REXX Execs zu nutzen. Der Dateierstellungsservice des ISPF (File Tailoring Service) stellt eine elegante Möglichkeit dar, JCL zu generieren. Auch dies wird im siebten Kapitel vorgestellt.

Fehlermeldungen und ABENDs sind im Kapitel acht erläutert. Im neunten Kapitel finden Sie technische Daten, die für die Performance von Bedeutung sind.

Wer kann dieses Buch nutzen?

Alle, die unter den Betriebssystemen MVS/XA und MVS/ESA arbeiten. Parameter, die erst mit MVS/ESA verfügbar wurden, sind mit (ESA) gekennzeichnet. Parameter, die mit der Version 4 von MVS/ESA verfügbar wurden, sind mit (ESA/4) gekennzeichnet.

Ob Sie JCL-Neuling sind oder schon JCL-Erfahrung besitzen, spielt keine Rolle. Sie werden in jedem Fall von diesem Buch profitieren können.

Was dieses Buch nicht ist

Dieses Buch will kein Ersatz für ein IBM Handbuch sein. Aus diesem Grund wird Ihnen immer nur die Syntax vorgestellt, die Sie in der Praxis brauchen. Auf eine vollständige Darstellung von in der Praxis nur sehr selten eingesetzten Parametern wird daher verzichtet.

1.1 Grundbegriffe

Wie Sie sehr bald merken werden, tauchen einige Begriffe in diesem Buch immer wieder auf. Es handelt sich dabei um zentrale Merkmale der MVS-Umgebung.

MVS-Betriebssysteme

MVS/ESA ist bereits die dritte Generation der MVS-Betriebssysteme. MVS steht für Multiple Virtual Storage, ein Betriebssystem, das in der Lage ist, einen virtuellen Speicher zu benutzen.

MVS ging aus einer Reihe von Vorgängersystemen hervor, deren Ursprung in das Jahr 1964 zurück reicht. Damals brachte IBM die Rechner der Reihe /360 auf den Markt und versprach seinen Kunden Aufwärtskompatibilität. Dieses Versprechen wurde bis heute immer eingehalten. Das wichtigste Betriebssystem der Nachfolgereihe /370 wurde MVS. In seiner Ursprungsversion MVS/370 bot es einen virtuellen Speicher mit 16 MB und eine 24-Bit Adressierung.

Schon gegen Ende der 70er Jahre konnte MVS/370 den gewachsenen Platzanforderungen nicht mehr gerecht werden. Insbesondere On-Line-Anwendungen wie IMS forderten ihren Preis. Die Antwort war das Betriebssystem MVS/XA, wobei XA für EXtended Architecture steht. MVS/XA arbeitet mit einem virtuellen Speicher von 2 GB und einer auf 31 Bit erweiterten Adressierung. Das 32. Bit gibt den Adressierungsmodus an. Um die Kompatibilität zu wahren, wurde der virtuelle Speicher logisch um die 16 MB Linie gruppiert. 24-Bit Programme laufen unterhalb dieser Linie, 31-Bit Programme unterhalb und oberhalb dieser Linie. Da unter MVS/XA auch die Abwicklung von I/Os neu gestaltet wurde, brachte dieses System erhebliche Vorteile gegenüber MVS/370.

Der Bedarf an virtuellem Speicher wuchs und wächst unaufhaltsam weiter. Moderne On-Line-Systeme wie DB2 oder CICS brauchen immer mehr Platz. Die zu verwaltenden Objekte nehmen an Größe zu. Das neueste Betriebssystem - MVS/ESA (Enterprise System Architecture) - bietet einen maximalen adressierbaren Bereich von fast unvorstellbaren 16 TB (Terabyte). Während beim Wechsel von MVS/370 auf MVS/XA die Adressierung von 24 Bit auf 31 Bit wuchs, ist bei MVS/ESA der adressierbare Bereich in die Breite gewachsen. Auch bei MVS/ESA wird mit 31 Bit adressiert. Zusätzlich zum normalen Adreßraum mit 2 GB, der schon unter XA verfügbar war, gibt es unter ESA Nur-Datenräume mit der Bezeichnung Data Space und HiPer Space, die ebenfalls 2 GB groß sind. Data Spaces sind byteadressierbar, in HiPer Spaces wird blockweise (4.096 Byte) adressiert. HiPer steht übrigens für High Performance.

Durch die Einführung von Nur-Datenräumen wird die Möglichkeit gegeben, die Daten adreßmäßig vom normalen Adreßraum zu trennen. Derzeit lassen sich die Data bzw. HiPer Spaces von höheren Programmiersprachen nur über die Data Windowing Services ansprechen, deren Benutzung derzeit noch etwas umständlich ist. Mit Assembler lassen sich beide Datenräume nutzen. Softwareprodukte wie DB 2 oder SYNCSORT nutzen diese Datenräume bereits. Dadurch ist es möglich, physische I/Os zu reduzieren.

Speichermedien

Für die Speicherung von Daten stehen Ihnen zwei verschiedene Möglichkeiten zu Verfügung:

1. Plattenspeicher mit der Möglichkeit des Direktzugriffs.
2. Kassetten und Bänder für die Speicherung von Massendaten und die Datensicherung mit der Möglichkeit des rein sequentiellen Zugriffs.

Der einzelne Datenträger, egal ob Platte oder Band/Kassette, wird als **Volume** bezeichnet. Jedes Volume hat eine eindeutige Bezeichnung, von der häufig auch als vol-ser-no die Rede ist. Diese Volume-serial-number dient der exakten Bezeichnung. Bei Platten enthält die vol-ser-no Buchstaben, die Angaben für Bänder/Kassetten sind rein numerisch.

Die **Unit** bezeichnet eine Gruppe gleichartiger Datenträger. Diese Bezeichnung wird Ihnen häufiger als die vol-ser-no begegnen, denn oft ist es praktischer, sich nur für einen Datenträgertyp zu entscheiden, als für eine ganz bestimmte Platte oder ein bestimmtes Band.

Plattenspeicher

Die am häufigsten genutzten Plattenspeicher sind die IBM Modelle 3380 und 3390. Die älteren Typen der Baureihe 3350 und 3330 sind heutzutage nur noch vereinzelt im Einsatz. Die einzelnen Baureihen unterscheiden sich hinsichtlich ihrer physischen Konfiguration. Innerhalb einer Modellreihe gibt es nur Unterschiede hinsichtlich der Gesamtspeicherkapazität.

Jeder Plattenspeicher hat eine **Volume**-Bezeichnung. Ein Platten-Volume besteht aus der Summe der übereinander angebrachten physischen Platten, die um eine gemeinsame Achse rotieren.

Im Zusammenhang mit der Speicherung von Daten auf Platte gibt es zwei wichtige Begriffe:

1. **Spur (Track)** und
2. **Cylinder**

Jede Platte besteht aus Magnetspuren, die in der Form von konzentrischen Kreisen auf der Plattenoberfläche abgelegt sind. Obwohl alle Spuren einen unterschiedlichen Radius besitzen, ist ihre Speicherkapazität gleich. Die Speicherdichte der einzelnen Spuren ist unterschiedlich und nimmt nach innen hin zu. Dies hat aber heute keinerlei praktische Bedeutung mehr.

Für jede Plattenoberfläche gibt es einen Lese-Schreibkopf. 15 Lese-Schreibköpfe sind an einen Zugriffsarm gekoppelt und werden immer gemeinsam über die Plattenoberflächen geführt. Die Summe der übereinander auf den verschiedenen Platten liegenden Spuren wird als **Cylinder** bezeichnet.

Kontrolliert werden die Plattenspeicher von den Storage Control Units (SCU). Verbreitet sind heutzutage die Baureihen 3880 und 3990. Die SCU der Baureihe 3990 hat die Besonderheit, daß sie Platten des Typs 3380 und 3390 gemeinsam kontrollieren kann. Die Tatsache, daß diese Plattentypen hinsichtlich der Spurkapazität unterschiedlich konfiguriert sind, macht der SCU nichts aus. Allerdings kann es Ihnen Probleme bereiten, in dieser Konfiguration eine optimale physische Recordgröße zu errechnen.

Die einzelnen Plattenbaureihen unterscheiden sich hinsichtlich der **Spurkapazität**. Diese Unterschiede sind für den Programmierer von großer Bedeutung. Ziel sollte es immer sein, den Platz auf der Platte optimal zu nutzen. Dies kann man dadurch erreichen, daß man es MVS ermöglicht, eine optimale physische Recordgröße zu errechnen. Man kann auch selber eine entsprechende Angabe mit dem BLKSIZE-Parameter machen, dann ist man aber auch selber für die Optimierung zuständig. Dies läßt sich durch Benutzung von JCL erreichen. Wer JCL falsch einsetzt, kann in diesem Punkt jedoch böse Überraschungen erleben. Im schlimmsten Fall kommt man auf Nutzungsgrade, die unter 10% des verfügbaren Platzes liegen.

Die folgende Übersicht zeigt die wesentlichen physischen Merkmale verschiedener Plattenspeicher:

Baureihe	Bytes/Spur	Spur/Cyl	Kap/Vol (MB)	Kap/Laufwerk (GB)
3380 A/B/D	47.476	15	630	2,52
3380 E/J	47.476	15	1.260	5,04
3380 K	47.476	15	1.890	7,56
3390-1	56.664	15	946	11,35
3390-2	56.664	15	1.892	22,7

Da mehrere physische Volumes zu einem logischen Volume zusammengefaßt werden können, ergeben sich maximale Speicherkapazitäten von bis zu 22,7 GB auf einer 3390-2.

Magnetbänder und Kassetten

Vornehmlich zum Zwecke der Datensicherung werden heutzutage Magnetkassetten und Magnetbänder verwendet, wobei die Magnetkassetten wegen ihrer leichteren Bedienung die

Magnetbänder weitgehend verdrängt haben. Daten auf Band oder Kassette können nur sequentiell verarbeitet werden. Die Speicherkapazität einer Kassette vom Typ 3480 liegt bei 250 MB.

Was die JCL betrifft, so macht es keinen Unterschied, ob Sie auf eine Kassette oder ein Band zugreifen. Aus diesem Grund werden in diesem Buch die Begriffe Band und Kassette synonym benutzt.

Logische versus physische Records

In Ihren Anwendungsprogrammen werden immer logische Records verarbeitet. Auf den Datenträgern werden aber die logischen Records nicht einzeln abgespeichert; sie werden zu physischen Records, die aus mehrerem logischen Records bestehen, zusammengefaßt. Diese physischen Records heißen Blöcke. Ihre Größe kann mit einem bestimmtem JCL-Parameter beeinflußt werden.

Zwischen den Blöcken müssen Lücken, sogenannte Gaps, frei bleiben. Dies ist notwendig, um dem Lese-Schreibkopf das Aufsetzen am Beginn eines Blocks zu ermöglichen. Die Größe der Lücke ist erheblich. Sie beträgt 492 Byte bei einer 3380 und 666 Byte bei einer 3390. Auch zwischen den Blöcken von Daten, die auf Band oder Kassette gespeichert werden, müssen Lücken frei bleiben.

Diese Gaps sind es, die das Abspeichern in Blöcken erforderlich machen. Würde man z.B. einen logischen Record von 80 Byte Länge in der physisch gleichen Länge abspeichern, so würde auf einen physischen Record von 80 Byte Länge eine Lücke von 492 Byte folgen. Der Effekt wäre eine miserable Platzausnutzung. Nur 1/6 der Spur wäre mit Daten belegt. Bei Bändern/Kassetten sieht diese Rechnung ähnlich aus.

Während Sie früher gezwungen waren, sich eine optimale Blockgröße selbst zu errechnen, wird diese Funktion heute vom Betriebssystem übernommen. Dies bedeutet für Sie eine wesentliche Erleichterung und garantiert sowohl eine optimale Speicherplatznutzung als auch eine gute I/O-Performance.

Datenverwaltung - VTOC, VVDS und Katalog

Ebenso wie die Betriebssysteme, so ist auch die Art und Weise, wie Datenbestände verwaltet werden, evolutionär gewachsen. Dies erklärt die duale Struktur der Datasetverwaltung. Die untere Ebene ist die des Plattenverzeichnisses (VTOC) und der Bandverzeichnisse (Label), die obere Ebene ist die des Kataloges.

Das Plattenverzeichnis wird als VTOC bezeichnet. Die Abkürzung bedeutet Volume Table of Content. Der VTOC enthält die Namen der auf der Platte befindlichen Datasets und gibt die Position auf der Platte an. Außerdem wird im VTOC Information über Umfang und Position von freiem Platz auf der Platte verwaltet.

Als Rechenzentren nur wenige Platten zu verwalten hatten, genügte es, sich die Platte zu merken, auf der ein Dataset angelegt wurde oder aber alle Platten durchsuchen zu lassen, wenn von einem Dataset gelesen werden sollte. Mit der Zunahme der Platten wurde diese

Form der Verwaltung immer ineffizienter. Hinzu kommt, daß das Durchsuchen des VTOCs eine sehr I/O-intensive Angelegenheit ist.

Um das Auffinden von Datasets zu vereinfachen, wurde eine Katalogstruktur entwickelt. In seiner ursprünglichen Form war der Katalog ein Dataset, in dem die DSNs aller anderen Datasets und der Name der Platte verzeichnet war, auf dem die Datasets tatsächlich standen. Um ein Arbeiten mit dieser Katalogstruktur zu ermöglichen, war es jetzt notwendig geworden, beim Anlegen eines Datasets darauf zu achten, daß ein Eintrag sowohl im VTOC der Platte als auch im Katalog erfolgte. Die Einträge müssen synchron gehalten werden.

In Nicht-SMS-Umgebungen gibt es immer noch die Möglichkeit, sogenannte nicht katalogisierte Datasets anzulegen. Für diese Datasets existiert zwar ein Eintrag im VTOC, aber kein Eintrag im Katalog.

Ein weiteres Problem kann auftreten, wenn ein Dataset angelegt wird, dessen Namen im Katalog schon existiert. Wird der Katalogeintrag vor dem Anlegen nicht entfernt, so wird das Dataset zwar im VTOC einer Platte eingetragen, nicht aber im Katalog. Auch hier ist das Ergebnis ein nicht-katalogisiertes Dataset. Auch dieses Problem kann nur in Umgebungen auftreten, in denen SMS nicht aktiv ist.

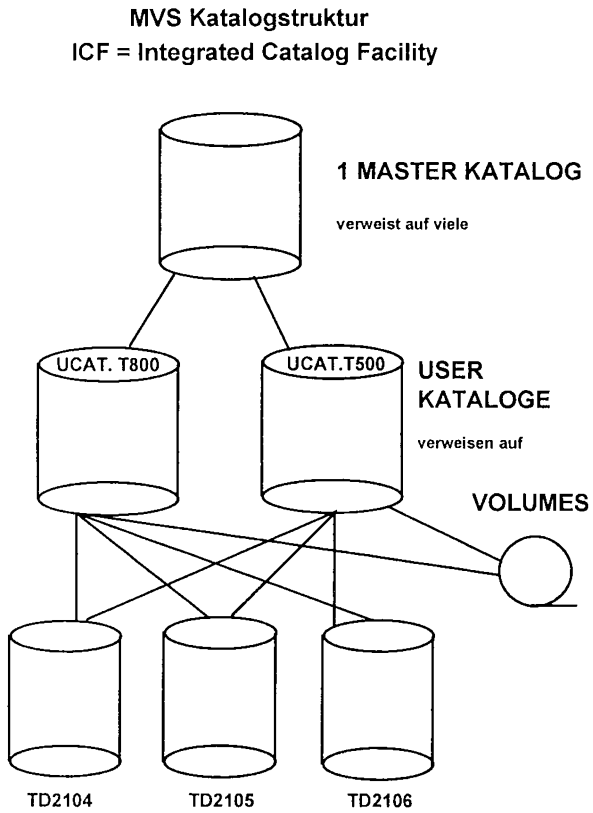
Eine Zeit lang existierten in MVS-Umgebungen verschiedene Katalogsysteme nebeneinander. Diesem Durcheinander wurde durch die Einführung der Integrated Catalog Facility (ICF) ein Ende gemacht. Mit ICF können sowohl normale (sequentielle) als auch VSAM-Datasets verwaltet werden.

Der ICF-Katalog besteht aus einem Master- und mehreren Userkatalogen. Der Master-Katalog enthält als Eintrag den ersten Teil des Datasetnamens (DSN) und den Verweis auf den Userkatalog. Im Userkatalog ist der eigentliche Datasetname (DSN), der Datenträger (UNIT) und die Plattennummer der Platte (VOLUME) verzeichnet, auf dem das gesuchte Dataset steht.

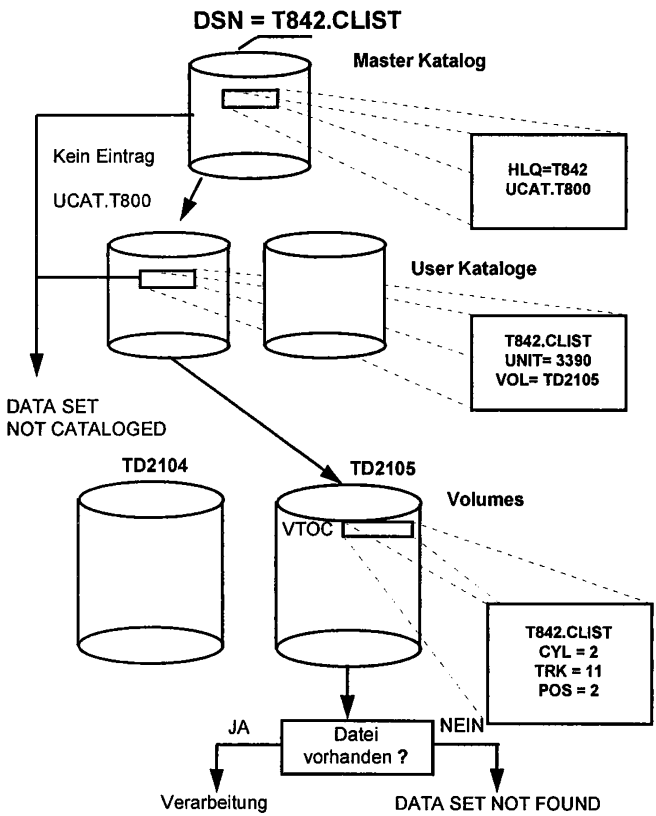
Warum nun diese scheinbar komplizierte Struktur? Ein Grund ist die Datensicherheit. Wie der Name schon vermuten läßt, ist der Master-Katalog ein zentraler Teil der Datasetverwaltung. Um ihn zu schützen, haben die normalen User auf diesem Katalog nur eine Leseberechtigung. Einträge dürfen nur von der Systemprogrammierung gemacht werden. Bevor ein neuer User sich zum ersten Mal zum ISPF anmelden kann, muß seine USERID im Master-Katalog eingetragen sein, ansonsten kann das ISPF Profildataset nicht angelegt werden, wodurch das Anmelden zum ISPF mißlingt.

Im ihm zugeordneten Userkatalog hat der User Updateberechtigung. Sie ist notwendig, um z.B. ein Dataset neu eintragen zu können. Die Tatsache, daß mehrere Userkataloge eingesetzt werden können, bietet den Vorteil, daß der einzelne Userkatalog nicht zu groß wird, was wiederum für einen schnellen Zugriff sehr wichtig ist.

Die folgende Abbildung zeigt den grundsätzlichen Aufbau von Master- und Userkatalog:



Am folgenden Beispiel soll erläutert werden, wie mit Hilfe des Katalogs das Dataset UWIN.CLIST lokalisiert wird.



In der JCL wird nur der Datasetname UWIN.CLIST mit Hilfe des DSN-Parameters angegeben. Zunächst wird mit Hilfe des ersten Teil des DSN UWIN auf dem Master-Katalog gelesen.

Eintrag im Master-Katalog für User UWIN und für HLQ T999:

HLQ	**UCAT**
T999	UCAT.T800
UWIN	UCAT.T800

Der Master-Katalog enthält als Einträge nicht nur USERIDS sondern auch andere High Level Qualifier. High Level Qualifier sind zulässige erste Teile des DSN. Ob Sie diese HLQ benutzen dürfen, hängt von Ihrer RACF-Berechtigung ab.

Der Eintrag im Master-Katalog verweist auf den Userkatalog, in dem der **komplette** DSN verzeichnet ist:

Eintrag im Userkatalog für das Dataset UWIN.CLIST:

DSN	UNIT	VOLUME
UWIN.CLIST	3390	DSK013

Der Eintrag besagt, daß das Dataset UWIN.CLIST auf einer Platte des Typs 3380 steht, die die Bezeichnung DSK013 trägt.

Im VTOC der Platte DSK013 ist der DSN UWIN.CLIST und die Position dieses Datasets auf der Platte verzeichnet.

Was wäre passiert, wenn ein nicht existierendes Dataset gesucht worden wäre? Nehmen wir an, Sie hätten sich verschrieben und nach dem Dataset UWINN.CLIST gesucht. Da für UWINN kein Eintrag im Master-Katalog existiert, käme es zu einem

```
DATASET NOT CATALOGED
```

Fehler. Hätten Sie nach UWIN.CLST gesucht, so hätte der Eintrag im Userkatalog gefehlt und es wäre zum gleichen Fehler gekommen.

Neben den Komponenten Master-Katalog, Userkatalog und VTOC gibt es noch ein weiteres wichtiges Systemdataset: das VVDS (VSAM Volume Data Set). Auf jeder Platte, auf der VSAM-Datasets stehen, gibt es ein VVDS. Mit dem VVDS wird die große Zahl von Statistikeinträgen gehandhabt, die VSAM-Datasets mit sich herumschleppen müssen. Das VVDS enthält alle Statistikeinträge für die auf der gleichen Platte stehenden VSAM-Datasets. Diese Statistiken sind ein wichtiger Indikator für das Performanceverhalten von VSAM-Datasets. Vor der Einrichtung des VVDS wurden all diese Statistikeinträge im Userkatalog verwaltet. Dies führte zu einer hohen Zahl von Zugriffen auf den Userkatalog und beeinträchtigte sowohl die Performance des Katalogs, der seinerseits die Struktur eines

VSAM-Dataset hat, wie auch die der VSAM-Datasets allgemein. Durch die Einrichtung des VVDS wird die Zahl der Zugriffe auf den Katalog vermindert und das Synchronitätsverhalten verbessert, denn das VSAM-Dataset und das zugehörige VVDS liegen ja immer auf demselben Volume.

Da der Master- und Userkatalog ebenfalls als VSAM-Datasets organisiert sind, gibt es auf den Platten, auf denen Master- und Userkataloge angesiedelt sind, jeweils ein VVDS.

Datenverwaltung und -sicherung: DFHSM und RACF

Die große Zahl der Datasets, die von einem Großsystem zu verwalten sind, zwingt zu einer optimalen Datenorganisation. So stehen auf den meisten MVS-Installationen zwei große Systemsoftwarepakete zur Verfügung, die dem Datenmanagement und dem Datenschutz (Zugriffsschutz) dienen.

DFHSM (Data Facility/Hierarchical Storage Manager) ist ein System, daß die Datenverwaltung auf dem Großsystem optimieren hilft. Es besteht aus einer Migrations- und einer Backupkomponente.

Die Migrationskomponente sorgt dafür, daß nicht oder selten benötigte Datasets entweder komprimiert oder auf Band/Kassette ausgelagert und erst bei Bedarf wieder geladen werden. Fragmentierte Datasets werden automatisch wieder zusammengefügt.

Die Backupkomponente zieht von allen Daten in regelmäßigen Abständen Kopien (Backups). Im Falle eines Datenverlusts können diese Backups zum Recovern von Datasets verwendet werden.

RACF (Ressource Access Control Facility) regelt den Zugang zum System (Anmeldevorgang) und den Zugriff auf Daten (Lese- und Schreibvorgänge). Es stellt sicher, daß nur autorisierte User Zugang zu den für sie bestimmten Datenbeständen haben. Zudem überwacht RACF die Art des Zugriffs.

Mehr über den DFHSM und RACF erfahren Sie in dem Band: *TSO* von Dr. Michael Teuffel (siehe Literaturverzeichnis im Anhang).

Datenmanagement - SMS

SMS - System Managed Storage oder Storage Management Subsystem - ist ein Produkt, daß die Verwaltung von Datasets auf Plattenspeicher vereinfachen soll. Es kann unter MVS/ESA eingesetzt werden. Die meisten MVS-Installationen setzen SMS als selbstverständliche Ergänzung ein.

Die wichtigste Neuerung unter SMS ist die Tatsache, daß es in einem von SMS kontrollierten Umfeld keine nicht-katalogisierten Datasets mehr geben kann. Außerdem lassen sich mit SMS VSAM-Datasets ohne die Benutzung der Utility IDCAMS definieren (und auch löschen!).

Bei SMS unterscheidet man die Zustände installiert und aktiviert. Bereits im nicht aktivierten Zustand bringt SMS einige erhebliche Vorteile. Wegen seiner großen Bedeutung wurde SMS das gesamte Kapitel 6 in diesem Buch gewidmet.

Zugriffsmethoden

Die Zugriffsmethoden erlauben es, in Ihrem Anwendungsprogramm logische Records verarbeiten zu können, ohne sich um deren physische Struktur kümmern zu müssen. Wir werden uns mit den zwei wichtigsten Zugriffsmethoden auseinandersetzen: VSAM und QSAM.

Die Zugriffsmethode ISAM (Indexed Sequential Access Method) ist veraltet und wurde durch VSAM abgelöst. Sie wird deshalb in diesem Buch nicht besprochen.

VSAM

VSAM ist die Abkürzung für Virtual Storage Access Method. Es ist eine Zugriffsmethode, die vier Datasetarten unterstützt, eine davon mit der Möglichkeit des Direktzugriffs. Das Thema VSAM ist so umfangreich, daß ihm das komplette Kapitel 5 gewidmet wurde.

VSAM-Datasets können *nur* auf Plattenspeichern angelegt werden.

QSAM

QSAM steht für **Queued Storage Access Method**. Es ist eine Zugriffsmethode, die den Zugriff auf sequentielle Datasets erlaubt. Die Datasets können auf Platte oder Band/Kassette residieren.

Dem Anwendungsprogramm wird im Hauptspeicher Platz für fünf Blöcke in einem sogenannten Puffer zur Verfügung gestellt. Die Größe dieser Puffer entspricht der Blocksize. Setzt Ihr Programm den OPEN-Befehl für ein sequentielles Dataset ab, so werden die fünf Puffer gefüllt. Ihr Programm arbeitet dann die Records sequentiell mit READ-Anweisungen ab.

Während Ihr Programm die Daten aus den Puffern verarbeitet, werden die bereits gelesenen Puffer asynchron wieder aufgefüllt. Sind die Daten aus dem fünften Puffer abgearbeitet worden, so kann unmittelbar im ersten Puffer weitergelesen werden, da QSAM die Daten schon zur Verfügung gestellt hat. Um die Tatsache, daß irgendwelche Puffer im Spiel sind, brauchen Sie sich in Ihrem Anwendungsprogramm nicht zu kümmern. Dies alles übernimmt die Zugriffsmethode QSAM.

Beim Schreiben wird mit QSAM zunächst in die Puffer geschrieben und dann asynchron auf den gewünschten Datenträger übertragen.

Record-Formate

Bei sequentiellen oder PO-Datasets werden folgende Record-Formate unterschieden:

1. Fixe logische Recordlänge (F)
2. Variable logische Recordlänge (V)
3. Undefinierte logische Recordlänge (U)
4. Mehrere logische Records sind zu einem physischen Record zusammengefaßt. Dies wird durch ein auf F oder V folgendes B dargestellt.

Das am häufigsten verwendete Format ist FB, wobei B für blocked steht. Die logischen Records werden zu einem physischen Block zusammengefaßt. Die Länge des physischen Blocks ist ein Mehrfaches der logischen Recordlänge. Die maximal zulässige Blockgröße ist 32760.

Sollen variabel lange logische Records zu einem Block zusammengefaßt werden, so ist als Format VB anzugeben. Am Anfang eines jeden Blocks befindet sich ein 4 Byte langes Feld, das die tatsächliche Länge des Blocks beschreibt. Dies hat Auswirkungen auf Ihr Anwendungsprogramm. So stehen Ihnen bei einer maximalen Blockgröße von 32760 Byte nur 32756 zur Verfügung.

Beim undefinierten Record-Format gibt es keine logischen Records. Dieses Format wird nur für Bibliotheken benötigt, in deren Member sich ausführbare Programme - Load Modules - befinden.

Datenorganisation - PS oder PO

Wenn Sie Batch-Verarbeitung betreiben, so haben Sie es am häufigsten mit Physisch Sequentiellen Datasets - PS-Datasets - zu tun. Bei dieser Datasetform werden die Records einer nach dem anderen weg geschrieben. In Jobs werden sehr häufig PS-Datasets als Ausgabedatasets erzeugt.

Daneben gibt es noch eine weitere Organisationsform, die Bibliothek oder PO-Dataset. PO kommt aus dem Englischen und heißt Partitioned Organized, meint also eine Datasetform, die in sich noch einmal untergliedert ist.

Der Vergleich mit der Bibliothek ist recht passend, denn wie eine Bibliothek besteht ein PO-Dataset aus einem Verzeichnis (Directory) und vielen Büchern (Member). Editiert werden die Member und nicht das gesamte Dataset. Bibliotheken werden u.a. zum Entwickeln von Programmen eingesetzt, aber auch die JCL, die einen Job steuert, wird als Member einer Bibliothek abgelegt. Das Haupteinsatzgebiet von Bibliotheken ist ISPF. In Jobs werden nur sehr selten Bibliotheken erzeugt.

1.2 Der Ablauf des Jobs

Ein Job ist eine Folge von JCL-Anweisungen. Jeder Job muß mit der JOB-Anweisung beginnen. Ein Job kann ein oder mehrere Programme ausführen. Für jedes Programm wird eine EXEC-Anweisung benötigt. Die Beziehung zu den physischen Daten wird über DD-Anweisungen hergestellt. DD steht für Data Definition. Damit haben wir die drei wichtigsten Arten von JCL-Anweisungen kennengelernt. Wie sie syntaktisch aussehen, wird später erläutert.

Ein Job wird als Member eines PO-Dataset abgelegt. Es ist möglich, mehrere Jobs in ein Member zu stellen. Diese Vorgehensweise ist jedoch unüblich. Ein kompletter Job kann folgendermaßen aussehen:

```
EDIT ---- UWIN.SCHULUNG.IDCAMS(BR1401) - 01.05 ----------------------------
COMMAND ===>                                                SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000001 //UWIN217C JOB (UWIN,ABTDG02),'M.WINTER',CLASS=X,MSGCLASS=H
000002 //BR14    EXEC PGM=IEFBR14
000003 //DELDS   DD   DSN=UWIN.TEST.OUTFILE,
000004 //             DISP=(OLD,DELETE)
****** **************************** BOTTOM OF DATA ****************************
```

Wenn Sie jetzt noch nicht alles verstehen, ist das nicht schlimm. Alles wird noch ausführlich erläutert. Die grundsätzliche Folge der verschiedenen Anweisungen ist jedoch gut zu erkennen.

Alles, was vom Beginn einer EXEC-Anweisung bis zum Beginn der nächsten EXEC-Anweisung steht, wird als Step oder Job Step bezeichnet. Ein Job muß zumindest einen Step besitzen. Er kann aus bis zu 255 Steps bestehen.

Die DD-Anweisungen gehören immer zum Step. Ihre Zahl ist auf 3273 pro Step limitiert. Mit Hilfe einer DD-Anweisung wird entweder ein Dataset erzeugt oder ein existierendes Dataset angesprochen.

Um den Job zu aktivieren, geben Sie in der Kommandozeile des ISPF-Bildschirms den Befehl: SUBMIT ein. Ihr Job durchläuft dann 5 Phasen, die jetzt im einzelnen besprochen werden.

Die 5 Jobphasen

Die Abbildung auf der folgenden Seite zeigt den Ablauf des Jobs:

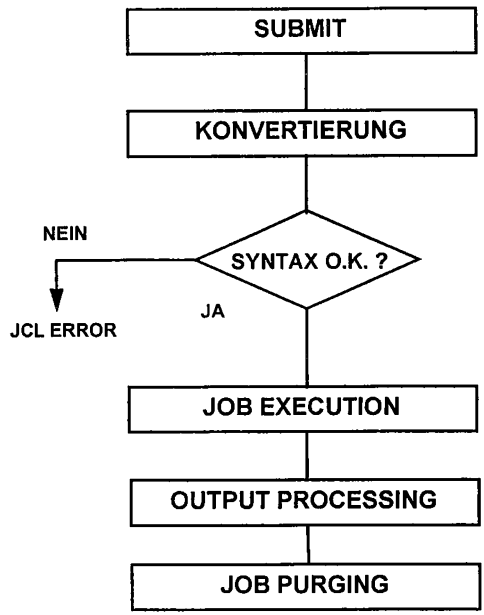
SUBMIT

Ihren Job erstellen Sie in der Regel unter ISPF als Member eines PO-Dataset. Wollen Sie den Job ausführen lassen, so setzen Sie den ISPF-Befehl: SUBMIT ab.

Durch den SUBMIT-Befehl übergeben Sie Ihren Job der Kontrolle des Job Entry Subsystems (JES). Von JES gibt es zwei Varianten: JES2 und JES3. Beide unterscheiden sich etwas in der Art, wie Sie Jobs abhandeln. JES2 ging aus einem Vorgängerprodukt namens HASP hervor. Dieses Houston Automatic Spooling Program wurde in den 60er Jahren für Einprozessorbetriebssysteme entwickelt. Die meisten MVS-Installationen nutzen heute JES2. JES3 wurde aus dem Vorgängerprodukt ASP (Asynchronous Multiprocessing System) entwickelt und war von Anfang an für Mehrprozessorumgebungen ausgelegt. Obwohl heute fast alle Mainframes mehrere Prozessoren haben, ist JES2 zu über 90% im Einsatz.

Die Operator-Meldungen, die Sie zurückbekommen, beginnen mit $HASP. Sie stammen also noch aus der Zeit, als dieses Job Management System entwickelt wurde.

Die fünf JOB-Phasen::



Konvertierung und Syntaxprüfung

Da JCL Variable enthalten kann (siehe dazu das Kapitel JCL-Prozeduren), müssen diese Variablen aufgelöst werden, bevor eine syntaktische Prüfung erfolgen kann.

Zusammen mit den JCL-Anweisungen können auch Daten übergeben werden. Diese sogenannten SYSIN-Daten nehmen am Konvertierungsprozeß nicht teil. Daher ist es Unsinn, dort Variable anzugeben, auch wenn dies wünschenswert wäre. Da SYSIN-Daten nicht auf Syntax geprüft werden, können sie auch keinen JCL Syntaxfehler auslösen.

Der Konvertierung folgt die Syntaxprüfung. Dort werden die JCL-Anweisungen auf formale Richtigkeit geprüft. Grob gesagt werden Fehler links vom Gleichheitszeichen fast immer erkannt. Fehler auf der rechten Seite des Gleichheitszeichens werden nur dann erkannt, wenn sogenannte Schlüsselwörter falsch angegeben wurden oder positionelle Parameter an der falschen Stelle stehen. Wenn Sie syntaktische Fehler begangenen haben, bekommen Sie eine:

```
JCL ERROR-
```

Meldung. Ihr Job kommt dann erst gar nicht auf die Warteschlange. Noch unter HASP war dies anders. Da konnte es passieren, daß man nach dem SUBMIT 30 Minuten warten mußte, bis der Job anlief und dann wegen eines trivialen Fehlers abbrach. Das kann heute nicht mehr passieren.

Logische Fehler werden beim Syntax Check nicht erkannt. So können Sie ein Programm aufrufen, das gar nicht existiert. Solange der Programmname syntaktisch korrekt eingegeben ist (max. 8 Zeichen, darf nicht mit einer Zahl beginnen), wird es nicht zu einem JCL-Fehler kommen. Wohl aber kommt es zu einem ABEND zur Laufzeit. Es kann aber während der Laufzeit des Programms zu einem Allokationsfehler kommen, der ebenfalls als JCL ERROR gemeldet wird. Syntaktische und Allokationsfehler sollten nicht miteinander verwechselt werden.

Die Ausführung des Jobs (Job Execution)

Wenn Ihr Job syntaktisch einwandfrei ist, dann wird er auf einer Warteschlange plaziert und steht zur Ausführung an. Die Frage, wann Ihr Job ausgeführt wird, hängt zuallererst von der Job-Klasse und deren Priorität ab. Die Frage des Zeitpunkts des SUBMITs ist nur dann interessant, wenn Jobs in der gleichen Klasse laufen.

Job-Klassen sind ein Instrument, um den Ablauf der Jobs zu steuern. Eine Job-Klasse kann einen oder mehrere Initiator besitzen. Der Initiator ist ein Systemprogramm, daß einen Job ausführt. Wie dies im einzelnen geschieht, wird noch erläutert.

Job-Klassen werden eingerichtet, um unterschiedlichen Bedürfnissen Rechnung zu tragen. So gibt es Job-Klassen mit einem hohen Anteil an CPU-Zeit, Job-Klassen, in denen Band/Kassettenverarbeitung zugelassen ist oder nicht oder Job-Klassen, die Jobs nur zu bestimmten Zeiten, z.B. nachts laufen lassen. Wichtig zu wissen ist, daß mit den Job-Klassen auch Limitierungen einhergehen. So gibt es in den meisten MVS-Installationen nur wenige Job-Klassen mit unlimitierter CPU-Zeit. Wenn aber die Job-Klasse ein CPU-Zeitlimit vorgibt, so können Sie es auch durch die Angabe entsprechender JCL-Parameter *nicht* erhöhen.

Die Ausgabeverarbeitung (Output Processing)

Wenn Ihr Job Output, z.B. Druckoutput erzeugt, so wird dieser nicht unmittelbar auf den Drucker gelegt, sondern wird in einem Zwischendataset gespeichert. Diese Vorgehensweise ist notwendig, da Sie in einer Multiprogramming Umgebung arbeiten.

Dieses Zwischendataset wird auf dem sogenannten Spool Volume angelegt. Dies geschieht automatisch. Diese Form des Outputs wird als SYSOUT (SYStem OUTput) bezeichnet. Erst wenn der Step beendet ist, wird mit dem Drucken begonnen.

Job entfernen (Purge Processing)

Ist der letzte Output, den der Job erzeugt hat, abgearbeitet (z.B. gedruckt), so wird der Job aus dem System entfernt. In den meisten MVS-Installationen wird der Job von einem

Output Management System übernommen. Typische Beispiele sind das BETA92 oder IOF (Input-Output Facility). Dort können Sie dann die Ergebnisse Ihres Jobs anschauen.

Der Initiator

Den Schritt Job-Ausführung wollen wir uns noch einmal genauer ansehen. Er ist der für uns entscheidende. Der Job wird wie bereits gesagt von einem Systemprogramm mit dem Namen Initiator zur Ausführung gebracht. Einzige Aufgabe des Inititiators ist es, nach Jobs Ausschau zu halten, die ausgeführt werden können.

Der Initiator ist einer oder mehreren Job-Klassen zugeordnet. Ein Job kann von einem Initiator nur dann ausgeführt werden, wenn die im Job angegebene Job-Klasse mit der übereinstimmt, für die der Initiator zuständig ist. Gibt es für eine bestimmte Job-Klasse nur einen Initiator, so kann das eine erhebliche Wartezeit bedeuten. Job-Klassen für Band- bzw. Kassettenverarbeitung besitzen meist nur einen oder wenige Initiator. Sehr aktiven Job-Klassen sind viele Initiator zugeordnet. Die Wartezeiten in diesen Klassen sind daher meist gering.

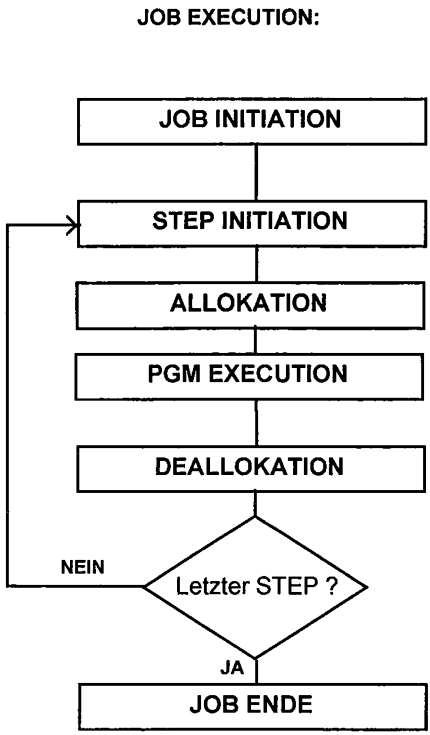
Job Initiation

Die wichtigste Aufgabe der Job Initiation ist sicherzustellen, daß im Job keine Datasets angefordert werden, die von anderen Jobs reserviert wurden. Sollte dies der Fall sein, so geht der Initiator in einen sogenannten 'wait state' und der Operator wird informiert. Ansonsten wird der nächste Schritt: Step Initiation ausgeführt.

Step Initiation

Wichtigste Aufgabe dieses Schrittes ist festzustellen, ob der Step wegen einer besonderen Bedingung unter Umständen nicht ausgeführt werden soll. Dazu wird, falls angegeben, der COND-Parameter überprüft bzw. Bedingungen geprüft, die mit IF/ELSE/ENDIF gesetzt wurden.

Der schematische Ablauf der Jobphase JOB-AUSFÜHRUNG sieht folgendermaßen aus:



Step Allokation

Durch die Allokation werden dem Step die benötigten Ressourcen zur Verfügung gestellt. Können die Ressourcen nicht zur Verfügung gestellt werden, so kommt es zum Abbruch des Steps (und damit des Jobs). Dies ist z.B. dann der Fall, wenn Sie für ein neues Dataset mehr Platz anfordern als zur Verfügung steht. In diesem Fall erhalten Sie einen JCL-Allokationsfehler. Es gibt auch Fälle, in denen der Step nicht anlaufen kann. Dieser Fall ist gegeben, wenn Sie mehr Lese-Schreibstationen anfordern als zur Verfügung stehen.

Die Art und Weise, wie die Allokation ausgeführt wird, unterscheidet sich je nachdem, ob bei Ihnen SMS aktiv ist oder nicht. Beim Aufsuchen vorhandener Datasets gibt es keinen Unterschied, wohl aber beim Anlegen neuer Datasets. Außerdem macht es bei der

Allokation einen Unterschied, wenn Sie Datasets auf Plattenspeicher oder Datasets auf Band/Kassette ansprechen.

Werden vorhandene Datasets, die auf Plattenspeicher residieren, angesprochen, so wird der Katalog abgesucht. Wird im Userkatalog der Eintrag für den DSN gefunden, so wird sichergestellt, daß die Platte, auf der sich das Dataset befindet, für den Step verfügbar ist. Zu diesem Zeitpunkt wird aber noch nicht in den VTOC der Platte geschaut. Dies geschieht erst, wenn das Programm einen OPEN absetzt.

Beim Anlegen von Nicht-SMS-Datasets wird eine Platte zugewiesen. Danach wird das Dataset angelegt, vorausgesetzt, der angeforderte Platz ist verfügbar. Das Dataset erhält dann einen Eintrag im VTOC. *Beachten Sie, daß das Dataset zu diesem Zeitpunkt nicht katalogisiert ist.* Reicht der Platz nicht, kommt es zu einem JCL-Fehler.

Werden SMS-Datasets angelegt, so wird zunächst überprüft, ob bereits ein Eintrag im Katalog für den zu vergebenden Datasetnamen existiert. Ist dies der Fall, so kommt es zu einem JCL-Fehler. Der Step bricht ab. Auf diese Weise wird verhindert, daß ein nicht-katalogisiertes Dataset angelegt wird. Sofern der Datasetname vergeben werden darf, wird eine Platte, die ebenfalls der Kontrolle von SMS unterliegt, zugewiesen. Das Dataset erhält einen VTOC-Eintrag und *wird sofort katalogisiert.* Dies ist der entscheidende Unterschied zu Nicht-SMS-Umgebungen. Dort wird das Katalogisieren von den Step-Termination-Routinen übernommen.

Wird anhand des Kataloges festgestellt, daß ein Dataset auf Band oder Kassette residiert oder soll ein neues Dataset auf Band/Kassette geschrieben werden, so erhält der Operator eine Meldung (Mount Message). Er muß dann eine freie oder die betreffende Kassette in eine Lese-Schreibstation einlegen. Die Lese-Schreibstation ist damit für Sie reserviert. Ein wichtiger Unterschied zwischen Plattenspeicher und Band/Kassette ist der, daß Ihnen die Lese-Schreibstation exklusiv zur Verfügung steht während sich viele User den Zugriff auf eine Platte teilen können ('shared access'). Beachten Sie auch, daß für **jedes** Dataset, das vom Band/von der Kassette gelesen oder geschrieben werden soll, **eine** Lese-Schreibstation belegt wird.

Die Ausführung des Programms

Nach der Allokation folgt die Programmausführung. Dabei können sowohl vom User geschriebene Programme wie auch Utility-Programme ausgeführt werden.

Erst wenn jetzt ein OPEN gemacht wird, wird beim Lesen von Plattendatasets der VTOC abgesucht. Sollte wider Erwarten das Dataset dort nicht vorhanden sein, kommt es zu einem

```
DATASET NOT FOUND
```

JCL Error.

Step-Ende (Step Termination)

In diesem Schritt werden die bei der Step-Allokation belegten Ressourcen wieder freigegeben. Außerdem wird versucht, die Datasets in den Zustand zu bringen, den der User mit Hilfe der JCL bestimmt hat. Dieser gewünschte Zustand kann sich unterscheiden, je

nachdem, ob der Step normal endet (normale Disposition) oder abbricht (abnormale Disposition).

Vor Stepbeginn vorhandene Datasets läßt man meist in dem Zustand zurück, wie man sie vorfand. Neue Datasets sollte man katalogisieren, wenn der Step normal endet. Bricht er ab, sollte man sie löschen.

Wichtig ist, daß bei neuen Datasets, die nicht SMS-kontrolliert sind, erst jetzt mit dem Versuch des Katalogisierens begonnen wird. Sollte der DSN im Userkatalog schon vorhanden sein, kann nicht katalogisiert werden. Der Step endet aber trotzdem normal. Das auf der Platte angelegte Dataset bleibt bestehen. Auf diese Weise ist dann ein sogenanntes nicht-katalogisiertes Dataset entstanden.

Sollten Datasets, die während des Programms geöffnet wurden, nicht geschlossen worden sein, so übernehmen die Step-Termination-Routinen diese Aufgabe.

Sollte der Job aus mehreren Steps bestehen, so werden jetzt die Schritte ab Step-Initiation erneut durchlaufen. Dies geschieht solange, bis der letzte Step beendet ist.

Das Ende des Jobs (Job Termination)

Normalerweise haben die Job-Termination-Routinen nichts zu tun. Es gibt aber einige Sonderfälle, in denen sie eingreifen müssen, z.B. dann, wenn ein Dataset an Folgestep übergeben wurde, dort aber nicht angenommen wurde. In diesem Fall entscheiden die Job Termination Routinen, was mit dem Dataset geschieht. Unter Umständen können dabei überraschende Resultate produziert werden.

1.3 Fehlermeldungen

Bei der Benutzung von JCL kann es zum Auftreten unterschiedlicher Fehlerbedingungen kommen. Es handelt sich dabei um JCL-Fehler und ABnormal ENDs (ABEND).

Bei JCL-Fehlern ist zwischen syntaktischen und Allokationsfehlern zu unterscheiden. Die syntaktischen JCL-Fehler lassen sich relativ leicht beheben. Sie werden beim Syntax Check entdeckt und gemeldet. Der Job kommt erst gar nicht zur Ausführung. Allokationsfehler treten erst während des Joblaufs auf. So kann es sein, daß der für ein neu anzulegendes Dataset angeforderte Platz nicht zur Verfügung gestellt werden kann. In diesem Fall kommt es zu einem SPACE REQUESTED NOT AVAILABALE-Fehler. Der Job Step bricht dann ab. Vorangegangene Steps sind aber bereits gelaufen. Der Job muß neu gestartet werden, und dieser Neustart kann wiederum mühsam sein. Allokationsfehler lassen sich nicht immer ausschließen.

ABENDs treten auf, wenn Ihr Programm versucht eine Instruktion auszuführen, die es - aus welchen Gründen auch immer - nicht ausführen darf. Meist ist es nicht einmal Ihr Programm selbst, daß die Kontrolle hat, sondern ein Systemprogramm, daß von Ihrem Programm mit Hilfe eines SVC (Supervisor Call) aufgerufen wurde. MVS beendet dann die Ausführung Ihres Programms und gibt einen sogenannten System Completion Code oder ABEND Code zurück. Der Code besteht aus drei Zeichen und ist mit dem SVC verknüpft. So zeigt ein Code von x13 ein Problem beim Öffnen eines Dataset an. Um die Ursache des

ABEND herauszufinden, müssen Sie das entsprechende IBM Manual: *GC28-1815 MVS/ESA Message Library: System Codes* konsultieren. Dort werden Sie häufig auf die Manuals: *MVS/ESA Message Library: System Messages Vol. I und II* verwiesen. Hier finden sich dann genaue Informationen zum ABEND und den zu ergreifenden Maßnahmen.

Im Kapitel 8 wird erläutert, wie die verschiedenen Fehlermeldungen zu interpretieren sind.

2 JCL

Bevor auf die einzelnen JCL-Anweisungen eingegangen wird, sollen an dieser Stelle die syntaktischen Grundzüge erläutert werden.

Aufbau der JCL-Anweisungen

Job Control-Anweisungen müssen in einem ganz bestimmten Format erstellt werden. Die Länge eines JCL-Records muß 80 Byte betragen. Diese Länge stammt noch aus der Lochkartenära, doch nur 72 Byte sind tatsächlich verfügbar.

Die JCL-Anweisungen werden in der Regel als Member einer Bibliothek gespeichert. Diese Bibliothek sollte den Namen:

`userid.*.CNTL`

tragen, wobei bei * ein beliebiger String bis zu acht Zeichen eingesetzt werden kann. CNTL steht als Abkürzung für Control.

Sie kann unter der ISPF Option 3.2 erstellt werden.

Eine JCL-Anweisung kann aus bis zu vier verschiedenen Feldern bestehen. Diese Felder werden im einzelnen beschrieben.

Das Identifikationsfeld

Das Identifikationsfeld beginnt auf Position 1 und hat eine Länge von 2 oder 3 Byte. Normale JCL-Anweisungen beginnen immer mit der Zeichenfolge '//'.

Besondere Bedeutung haben die Zeichenfolgen '/*', die das Ende von Daten oder Kontrollanweisungen kennzeichnet (Standard Delimiter Anweisung), und '//*', die am Beginn einer Kommentarzeile steht.

Beispiele:

ID-Feld	Bedeutung
//	Normale JCL-Anweisung
//*	Kommentarzeile
/*	Ende von Daten

Das Namefeld

Unmittelbar auf das Identifikationsfeld folgt das Namefeld. Der Name darf maximal acht Zeichen lang sein. Er darf aus den Buchstaben A-Z, 0-9 und den Zeichen '#', '@' und '$' bestehen. Nummern an der ersten Stelle des Namens sind nicht erlaubt.

Im Namefeld muß eine Angabe stehen, wenn es sich um eine JOB-Anweisung handelt. In einer EXEC-Anweisung sollte eine Angabe erfolgen. Dieser Name wird dann der sogenannte Stepname. *Ein* Step ist immer mit *einer* EXEC-Anweisung verbunden.

Bei DD-Anweisungen (Data Definition) wird mit der Angabe im Namefeld die Verbindung zwischen logischen und physischen Datasetnamen hergestellt.

Der Name selbst ist weitgehend wahlfrei mit Ausnahme bestimmter Namen, auf die später eingegangen wird. Grundsätzlich sollte man bei DD-Anweisungen keine Feldnamen vergeben, die mit SYS anfangen, denn es gibt eine Reihe reservierter Feldnamen.

Beispiele:

ID - Namefeld

```
//UWINUEB

//STEP1

//EINGABE1
```

Das Operationsfeld

Das Operationsfeld kennzeichnet die Funktion, die die JCL-Anweisung hat. Zwischen dem Namefeld und dem Operationsfeld muß mindestens ein blank stehen. Ausführlich betrachtet werden JOB, EXEC, DD und OUTPUT. IF, ELSE, ENDIF, SET und JCLLIB sind ab der Version 4.2 von MVS/ESA verfügbar. PROC und PEND haben nur für JCL-Prozeduren Bedeutung und werden im Kapitel 3: JCL-Prozeduren ausführlich vorgestellt.

Beispiele:

ID - Namefeld	Operation
//UWINUEB	JOB
//STEP1	EXEC
//EINGABE1	DD

Das Parameterfeld

Das Parameterfeld ist das umfangreichste von allen. Zwischen dem Operationsfeld und dem Namefeld muß mindestens ein Leerzeichen stehen. Besteht das Parameterfeld aus mehreren Parametern, so werden diese durch Kommata abgetrennt. Besteht ein Parameter aus

mehreren Subparametern, so müssen diese umklammert und mit Kommata abgetrennt werden, wobei keine Leerzeichen eingefügt werden dürfen (ein Leerzeichen kennzeichnet das Ende einer JCL-Anweisung).

Parameter können Schlüsselwort-Parameter oder Positionelle Parameter sein.

Beispiel 1 (ein Parameter im Parameterfeld):

```
ID Namefeld     Op   Parameter
//EINGABE       DD   SYSOUT=*
```

Beispiel 2 (mehrere Parameter im Parameterfeld):

```
ID-Namefeld     Op   Parameter
//EINGABE       DD   DSN=UWIN.TESTFILE,DISP=SHR
```

Beispiel 3 (ein Parameter mit mehreren Subparametern):

```
ID-Namefeld     Op   Parameter
//EINGABE       DD   DCB=(RECFM=FB,LRECL=287)
```

Das Kommentarfeld

Eine Kommentarzeile muß mit //* beginnen. Daran anschließend kann eine beliebige Zeichenfolge eingegeben werden. Die Benutzung von Kommentaren ist sehr nützlich für Dokumentationszwecke. Nicht kommentierte JCL ist genauso fürchterlich wie unkommentierte Programme. Nutzen Sie deshalb die Kommentarmöglichkeit.

Kommentare können auch rechts angehängt werden. Sie müssen von den Anweisungen durch mindestens ein Leerzeichen getrennt werden.

Fortsetzungsregeln

Ein Komma muß am Ende der ersten Zeile stehen, in jeder Folgezeile muß nach dem '//' ein Leerzeichen stehen. Neue Anweisungen müssen zwischen Spalte 4 und Spalte 16 beginnen. Wird auf Spalte 17 oder später begonnen, betrachtet MVS dies als Kommentar.

Hier ein Beispiel für Fortsetzungsregeln und Kommentare. Die JCL-Anweisung auf Zeile 300 wird fortgesetzt. Auf den Zeilen 300 und 400 stehen Kommentare. Auf Zeile 600 folgt den JCL-Anweisungen ein Kommentar:

```
EDIT ---- UWIN.SCHULUNG.CNTL(UEBUNG1) - 01.00 -------------- COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
000100 //UWINUEB1 JOB (998,ABTDG02),'M.WINTER',CLASS=X,MSGCLASS=H
000200 //STEP1  EXEC PGM=IEFBR14
000300 //* DIES IST EIN BEISPIEL FUER EINEN KOMMENTAR AUSSER-
000400 //* HALB VON JCL-ANWEISUNGEN
000500 //INPUT  DD DSN=UWIN.SCHULUNG.INPUT,
000600 //       DISP=SHR        KOMMENTARE KOENNEN AUCH HIER BEGINNEN
****** ***************************** BOTTOM OF DATA ****************************
```

Default-Werte

Für einige Parameter und Subparameter existieren systemseitige Default-Werte (Standardeinstellungen). Will man diese übernehmen, so braucht man den entsprechenden Parameter nicht zu kodieren, wenn es sich um einen Schlüsselwort-Parameter handelt. Bei positionellen Parametern muß die Abwesenheit desselben durch ein Komma gekennzeichnet werden. Wenn man es aber aus Dokumentationsgründen für erforderlich hält, so kann man Default-Parameter dennoch kodieren.

Parameter

Wie bereits erwähnt gibt es zwei Arten von Parametern:

1. **positionelle** (positional) und
2. **Schlüsselwort-** (keyword) Parameter

Wie der Name schon sagt, werden positionelle Parameter anhand ihrer Position erkannt. Werden sie weggelassen, so muß an ihre Stelle ein Komma treten. In diesem Fall werden Default-Werte angenommen. Gegebenenfalls sind mehrere Kommata für mehrere fehlende positionelle Parameter zu setzen. Fehlende positionelle Parameter am Schluß einer Anweisung brauchen nicht durch ein Komma ersetzt zu werden.

Schlüsselwort-Parameter werden unabhängig von ihrer Position erkannt. Sie können überall innerhalb des Parameter-Feldes stehen.

2.1 Die JOB-Anweisung

Ein Job ist eine Folge von JCL-Anweisungen, die mindestens aus einer JOB und einer oder mehreren EXEC-Anweisungen besteht. Jede EXEC-Anweisung markiert den Beginn eines Steps. Jeder Step ist mit der Ausführung eines Programmes verknüpft. Ein Job besteht also aus mindestens einem Step, er kann aber auch eine Vielzahl von Steps umfassen (bis 255).

Jeder Job muß mit einer Job-Anweisung beginnen. Aus ihm ergibt sich der Job-Name, der Abrechnungscode, die Klasse, in der der Job laufen soll, Art und Umfang der Nachrichten,

die man von MVS zurückbekommen will und der Ort, an den die Ausgaben zu schicken sind.

Syntax:

```
JOB (acct.info[,add.acct.info])[,'programmierer name'],
    CLASS=jobklasse,
    MSGCLASS=outputklasse[,]
    [NOTIFY=userid,]
    [MSGLEVEL=(anweisung,nachricht),]
    [TYPRUN=SCAN,]
    [REGION=wertK oder wertM,]
    [TIME=(min,sec),]
    [COND=(zahl,operator),]
    [RESTART=stepname]
```

Die einzelnen Parameter werden in den folgenden Kapiteln erläutert.

Der Beginn der JOB-Anweisung

Syntax:

```
//jobname    JOB (acct.info[,add.acct.info]),
             ['programmierer name',]
```

jobname ist ein bis zu 8 Zeichen langer String, der aus Buchstaben, Zahlen und den Zeichen '#', '@' und '$' bestehen kann. Das erste Zeichen darf keine Ziffer sein.

In jeder MVS-Installation gibt es eigene Regeln für die Vergabe von Job-Namen. Erkundigen Sie sich, wie die entsprechenden Vorschriften bei Ihnen sind. In den Beispielen wird davon ausgegangen, daß der Job-Name aus der USERID: UWIN und vier wahlfreien Zeichen bestehen darf.

Auf das JOB-Schlüsselwort folgt in Klammern die Abrechnungsinformation (**account info**). Auf sie kann das Feld **additional account info** folgen. Beide Angaben sind rein installationsabhängig. Erkundigen Sie sich, wie diese Angaben für Sie lauten müssen.

Der Name desjenigen, der den Job abschickt, folgt nach den Abrechnungsinformationen. Beachten Sie, daß der Name in Hochkommata gesetzt wird. (Das ist unumgänglich, wenn z.B. Punkte im Namen verwendet werden).'

Die folgenden Beispiele zeigen den Beginn von Job-Anweisungen.

```
EDIT ---- UWIN.SCHULUNG.CNTL(UEB1) - 01.00 ------------------ COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
000001 //UWINUEB1 JOB (998,ABTDG02),'M.WINTER',
000002 //          MSGCLASS=Y,
000003 //          CLASS=A
000004 //*
```

```
EDIT ---- UWIN.SCHULUNG.CNTL(UEB1) - 01.00 ------------------ COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
000001 //PRODP001 JOB (124),'M.WINTER',
000002 //          MSGCLASS=Y,
000003 //          CLASS=A
000004 //*
```

CLASS

Syntax:

CLASS=jobklasse

Als Wert für jobklasse sind 0-9 und A-Z zulässig.

Die Zuteilung von Jobs in bestimmte Job-Klassen dient der optimalen Auslastung des Systems. Es gibt daher Klassen für Produktion und Entwicklung, Klassen für Jobs mit hohem CPU-Anteil oder Klassen für Jobs, die viele I-Os durchführen. Auch die Frage, ob Band- bzw. Kassettenverarbeitung zugelassen ist, spielt bei der Wahl der Job-Klasse eine Rolle.

Erkundigen Sie sich, welche Job-Klassen in Ihrer Installation generiert sind und welche Sie benutzen dürfen.

MSGCLASS

Syntax:

MSGCLASS=outputklasse

Als Wert für outputklasse sind 0-9 und A-Z zulässig.

Der Parameter MSGCLASS teilt MVS mit, wo der Output hingestellt werden soll, d.h. soll er gedruckt werden oder nur ins IOF Menü laufen, wo er angeschaut werden kann.

Mit jeder Output-Klasse ist ein Maximum an sogenanntem SYSOUT Output verbunden. Wird dieses Maximum überschritten, so erhält man an einen S722 ABEND. Wenn dieser