



z/OS JCL and Utilities

(Course code ES07)

Student Notebook

ERC 4.0

Authorized

IBM | Training

Trademarks

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

BookManager®	CICS®	DB™
DB2®	DS6000™	DS8000®
ESCON®	FICON®	FlashCopy®
IMS™	Magstar®	MVS™
OS/390®	Parallel Sysplex®	RACF®
RETAIN®	S/390®	Solid®
System z®	VTAM®	z/OS®
z/VM®	z/VSE®	zEnterprise™
zSeries®	z10™	z9®
400®		

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

August 2011 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Contents

Trademarks	xi
Course description.....	xiii
Agenda	xv
Unit 1. Introduction to JCL	1-1
Unit objectives	1-2
z/OS basic skills information center	1-3
JCL course	1-5
Processors	1-6
Input/output devices	1-7
Operating systems	1-8
Programs	1-9
Data	1-10
Job control language	1-11
JES responsibilities	1-13
JCL-related actions	1-14
JCL statements	1-15
JOB statement	1-16
EXECUTE statement	1-17
Data definition statement	1-18
Input/output data	1-20
Multistep job	1-21
JCL errors	1-22
Program abnormal termination	1-23
Return codes	1-24
JES2/JES3 control statements	1-25
Checkpoint (1 of 2)	1-26
Checkpoint (2 of 2)	1-27
Unit summary	1-28
Unit 2. JOB, EXEC, and DD statements	2-1
Unit objectives	2-2
JCL statement format	2-3
Parameters	2-5
The JOB statement	2-7
JOB statement syntax	2-8
CLASS parameter	2-10
MSGCLASS parameter	2-11
MSGLEVEL parameter	2-12
NOTIFY and TYPRUN parameters	2-13
REGION parameter	2-14
MEMLIMIT parameter	2-16

LINES parameter	2-17
RESTART parameter	2-18
JOB statement examples	2-20
The EXEC statement	2-21
EXEC statement syntax	2-22
Program execution	2-23
PARM continuation and limit	2-24
TIME parameter	2-25
TIME parameter example	2-26
EXEC statement examples	2-27
The DD statement	2-28
Why data definition?	2-29
Accessing a data set	2-31
Data set concatenation	2-32
Data in the input stream	2-33
SYSOUT processing	2-35
COMMENT statement	2-37
Checkpoint (1 of 3)	2-38
Checkpoint (2 of 3)	2-39
Checkpoint (3 of 3)	2-40
Unit summary	2-41
 Unit 3. DD parameters: A second look.....	 3-1
Unit objectives	3-2
DD statement syntax	3-3
Permanent data set naming	3-4
Unit selection	3-5
Volume specification	3-7
DASD volume table of contents	3-8
DSCB content (format 1)	3-9
Space specification	3-10
Release unused DASD space	3-12
DISP: Syntax and defaults	3-13
DISP parameter	3-14
DISP processing: NEW	3-15
DISP processing: OLD	3-16
DISP processing: KEEP	3-17
DISP processing: CATLG	3-18
DISP processing: DELETE	3-19
DISP: Data set sharing	3-20
DISP processing: MOD	3-21
DISP processing: PASS	3-22
Backward reference: Example	3-24
Temporary (work) data sets	3-25
Temporary data sets: Example	3-26
Special DD statement parameters: DUMMY	3-27
Special DD names	3-28
Special DD statements: JOBLIB	3-29

Special DD statements: STEPLIB	3-30
To create a cataloged data set	3-31
To create a noncataloged data set	3-32
To reference an existing cataloged data set	3-33
To reference an existing noncataloged data set	3-34
Job logs	3-35
Job stream	3-36
Output from first run	3-37
Output from second run (1 of 3)	3-38
Output from second run (2 of 3)	3-39
Output from second run (3 of 3)	3-40
Output from third run (1 of 3)	3-41
Output from third run (2 of 3)	3-42
Output from third run (3 of 3)	3-43
Printed output from step 3	3-44
Checkpoint (1 of 3)	3-45
Checkpoint (2 of 3)	3-46
Checkpoint (3 of 3)	3-47
Unit summary	3-48
Unit 4. Introduction to utilities and conditional execution	4-1
Unit objectives	4-3
z/OS utilities	4-4
Classes of utilities	4-5
Utility selection	4-6
Utility control statement format	4-7
IEFBR14 allocation	4-8
IEBGENER card-to-disk	4-9
IEBGENER copy/print	4-10
IEBCOPY copy	4-11
IEBPTPCH print	4-12
IEBPTPCH print output	4-13
IEHLIST LISTVTOC	4-14
LISTVOC output	4-16
IDCAMS: DELETE	4-17
Conditional execution of JOB steps	4-18
Return code setting	4-19
JOB return code	4-20
COND parameter on the JOB statement	4-21
COND parameter on the EXEC statement	4-22
Abnormal termination testing	4-23
COND testing example	4-24
Checkpoint (1 of 2)	4-25
Checkpoint (2 of 2)	4-26
Unit summary	4-27
Unit 5. Data management, organization, and format.	5-1
Unit objectives	5-2

Record formats	5-3
Data control block parameter	5-4
Fixed-length record format	5-6
Variable-length record format	5-7
Variable block spanned format	5-8
Undefined-length record format	5-9
Control characters	5-10
Space versus blocking	5-11
System-determined block size	5-12
The DCB merge	5-13
Data set types supported	5-14
Data set organizations	5-16
Sequential organization	5-18
Partitioned organization	5-19
PDS space specification	5-20
Partitioned organization: Retrieval	5-21
Partitioned organization: Addition	5-22
Direct organization	5-23
Creating a data set: Decisions	5-24
SMS: Automating storage management policies	5-25
Constructs and ACS routines	5-26
SMS DD parameters	5-27
Space allocation: AVGREC	5-28
Assigning a data class	5-29
Data class list	5-30
SMS-managed data sets	5-31
Assigning a storage class	5-32
Partitioned organization: PDSE	5-33
Differences: PDS versus PDSE	5-34
HFS and zFS data sets	5-35
File system structure	5-36
Path name and file name	5-38
HFS data set characteristics	5-39
zFS file systems	5-40
HFS data set	5-41
JCL support	5-42
Allocate HFS: JCL example	5-43
Allocate and format a zFS aggregate	5-44
JCL example with a UNIX file	5-45
VSAM data sets	5-46
VSAM: RECORG and KEYOFF	5-48
Changes to job processing	5-49
Checkpoint (1 of 4)	5-50
Checkpoint (2 of 4)	5-51
Checkpoint (3 of 4)	5-52
Checkpoint (4 of 4)	5-53
Unit summary	5-54

Unit 6. Generation data groups.....	6-1
Unit objectives	6-2
Generation data groups: The need	6-3
GDG: DSNAME specification	6-4
GDG DSNAME: Base catalog entry	6-6
GDG DSNAME: Catalog entry	6-8
GDG example: First generation	6-9
GDG example: Second generation	6-10
GDG example: Third generation	6-11
GDG example: Limit exceeded	6-12
GDG ROLLIN and ROLLOFF	6-13
GDG in a multistep job	6-15
Checkpoint	6-17
Checkpoint (2 of 2)	6-18
Unit summary	6-19
Unit 7. Procedures	7-1
Unit objectives	7-2
Procedures (1 of 2)	7-3
Procedures (2 of 2)	7-4
Instream procedure	7-5
Cataloging a procedure	7-7
Procedure modification	7-9
Modifying EXEC statements	7-10
Modify EXEC statement syntax	7-11
Modifying DD statements	7-13
Modify DD statement syntax	7-14
Modify DD: Example (1 of 2)	7-15
Modify DD: Example (2 of 2)	7-16
Modifying procedures with symbolic parameters	7-17
Symbolic parameters: Example 1 (1 of 2)	7-19
Symbolic parameters: Example 1 (2 of 2)	7-20
Procedure ABCPROC	7-21
Checkpoint (1 of 3)	7-23
Checkpoint (2 of 3)	7-24
Checkpoint (3 of 3)	7-25
Unit summary	7-26
Unit 8. More about utilities.....	8-1
Unit objectives	8-2
IEBGENER copy	8-3
IEBCOPY copy	8-4
IEBCOPY listing	8-6
IEHLIST LISTPDSE	8-7
LISTPDS listing	8-9
IEHPROGM SCRATCH/RENAME	8-10
IEHPROGM listing	8-11
IEBUPDATE update in place	8-12

IEBUPDTE print output	8-13
The BPXBATCH utility	8-14
Checkpoint (1 of 2)	8-15
Checkpoint (2 of 2)	8-16
Unit summary	8-17
Unit 9. More on procedures	9-1
Unit objectives	9-2
Procedures and INCLUDE groups	9-3
Accessing the PROC: JCLLIB statement	9-5
The INCLUDE statement (1 of 2)	9-6
The INCLUDE statement (2 of 2)	9-8
Nested example	9-9
SET statement	9-10
Overriding symbolic parameters	9-11
Symbols and JCL	9-12
Checkpoint (1 of 2)	9-14
Checkpoint (2 of 2)	9-15
Unit summary	9-16
Unit 10. Selected JCL topics	10-1
Unit objectives	10-2
Conditional execution of JOB steps	10-3
Relational expressions	10-4
IF/THEN/ELSE/ENDIF statement construct	10-7
IF/THEN/ELSE/ENDIF example	10-9
JCL IF condition	10-10
Nested example	10-11
SYSOUT processing: Simple form control	10-12
FREE and SPIN parameters	10-13
SEGMENT parameter	10-15
SYSOUT DD statement limitations	10-16
OUTPUT statement: Setting defaults	10-18
OUTPUT statement: Explicit reference	10-20
SYSOUT distribution parameters	10-21
OUTDISP parameter	10-23
Checkpoint (1 of 2)	10-24
Checkpoint (2 of 2)	10-25
Unit summary	10-26
Unit 11. SORT and MERGE	11-1
Unit objectives	11-2
SORT process	11-3
EBCDIC collating sequence	11-4
Control statement format	11-5
SORT control statement	11-6
SORT on two fields	11-7
SORT JCL statements	11-8

SORT JOB example	11-9
MERGE process	11-10
MERGE control statement	11-11
MERGE JCL statements	11-12
MERGE JCL example	11-13
SORT/MERGE considerations	11-14
Another SORT example	11-16
Checkpoint (1 of 2)	11-17
Checkpoint (2 of 2)	11-18
Unit summary	11-19
 Unit 12. Multivolume and tape allocation.....	12-1
Unit objectives	12-2
Unit and volume syntax	12-3
Multivolume DASD	12-5
Multivolume: Example	12-6
Multivolume: Basic rules	12-7
Multivolume allocation	12-8
Multivolume: Allocation examples	12-9
Tape processing	12-10
The tape environment	12-11
Tape capacity	12-13
Tape unit selection	12-15
Tape data class	12-16
Magnetic tape-label parameter	12-18
Tape label format	12-19
IBM standard tape labels: Content	12-20
Unlabeled tapes	12-21
Bypass label processing	12-22
Tape allocation: Examples	12-23
Tape mounting: Retain	12-24
Unit affinity	12-25
Tape unit allocation: FREE=CLOSE	12-26
Checkpoint	12-27
Unit summary	12-28
 Unit 13. ABENDs.....	13-1
Unit objectives	13-2
ABENDs	13-3
ABEND recovery	13-4
LookAt on the Internet	13-5
LookAt for messages and abend codes	13-6
Common ABEND exercise	13-7
JCL STREAM	13-8
S222 ABEND (1 of 2)	13-9
S222 ABEND (2 of 2)	13-10
S806-04 ABEND (1 of 3)	13-11
S806-04 ABEND (2 of 3)	13-12

S806-04 ABEND (3 of 3)	13-13
S013-14 ABEND (1 of 4)	13-14
S013-14 ABEND (2 of 4)	13-15
S013-14 ABEND (3 of 4)	13-16
S013-14 ABEND (4 of 4)	13-17
S013-18 ABEND (1 of 4)	13-18
S013-18 ABEND (2 of 4)	13-19
S013-18 ABEND (3 of 4)	13-20
S013-18 ABEND (4 of 4)	13-21
Good run (1 of 4)	13-22
Good run (2 of 4)	13-23
Good run (3 of 4)	13-24
Good run (4 of 4)	13-25
Unit summary	13-26
Appendix A. Checkpoint solutions	A-1
Appendix B. JCL examples.....	B-1
Bibliography	X-1

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

BookManager®	CICS®	DB™
DB2®	DS6000™	DS8000®
ESCON®	FICON®	FlashCopy®
IMS™	Magstar®	MVS™
OS/390®	Parallel Sysplex®	RACF®
RETAIN®	S/390®	Solid®
System z®	VTAM®	z/OS®
z/VM®	z/VSE®	zEnterprise™
zSeries®	z10™	z9®
400®		

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

Course description

z/OS JCL and Utilities

Duration: 4.5 days

Purpose

This course is designed to teach you how to use z/OS job control language (JCL) and selected z/OS utility programs in an online batch environment. Both Storage Management Subsystem (SMS) and non-SMS JCL are discussed. Machine lab exercises complement the lecture material.

Audience

This course is intended for people who want to use z/OS JCL and utilities.

Prerequisites

Basic knowledge of information systems (IS) technologies. Students should be familiar with z/OS concepts and how these systems support the Enterprise servers. This knowledge can be obtained by attending course ES05, *An Introduction to the z/OS Environment*.

Objectives

After completing this course, a student should be able to:

- Code basic JCL statements using proper syntax and coding rules, including JCL for:
 - Creating new data sets
 - Referencing existing data sets
 - Condition code testing
 - IF/THEN/ELSE/ENDIF constructs
 - Generation data groups
 - Output routing
 - JCL enhancements introduced by various releases of Multiple Virtual Storage (MVS), OS/390, and z/OS
- Identify Storage Management Subsystem requirements

- Code instream and cataloged procedures
- Use symbolic parameters in procedures
- Code procedure overrides and additions super
- Use selected utility programs
- Describe tape processing facilities
- Code sort and merge control statements and associated JCL statements
- Recognize and resolve common abnormal terminations (ABENDs)

Curriculum relationship

This course is recommended prior to taking *z/OS Facilities* (ES15).

Other z/OS courses

- ES15, *z/OS Facilities*
- ES20, *z/OS System Services Structure*
- ES26, *SMP/E for z/OS Workshop*
- SS83, *z/OS VSAM and Access Method Services*
- ES52, *Writing REXX Programs for z/OS*
- ES41, *z/OS Installation*

Agenda

Day 1

- Welcome
- Unit 1: Introduction to JCL
- Unit 2: JOB, EXEC, and DD statements
- Unit 3: DD parameters: A second look
- Lab

Day 2

- Lab review: Day 1
- Unit 3: DD parameters: A second look (continued)
- Unit 4: Introduction to utilities and conditional execution
- Unit 5: Data management, organization, and format
- Lab

Day 3

- Lab review: Day 2 lab
- Unit 5: Data management, organization, and format (continued)
- Unit 6: Generation data groups
- Unit 7: Procedures
- Unit 8: More about utilities
- Lab

Day 4

- Lab review: Day 3 lab
- Unit 8: More about utilities (continued)
- Unit 9: More on procedures
- Unit 10: Selected JCL topics
- Unit 11: SORT and MERGE
- Lab

Day 5

- Lab review: Day 4 lab
- Unit 12: Multivolume and tape allocation
- Unit 13: ABENDs
- Optional lab

Unit 1. Introduction to JCL

What this unit is about

Job control tasks are needed to enter jobs into the operating system, control the system processing of jobs, and request the resources needed to run jobs. To perform these tasks, programmers code job control statements; these statements are referred to as job control language (JCL).

For your program to execute on the processor, the JCL statements must be correctly coded and placed in proper sequence in the job stream. The purpose of the job (JOB), execute (EXEC), and data definition (DD) statements are discussed in the unit.

The JCL errors, abnormal terminations (ABENDS), and return codes are also presented in this unit.

What you should be able to do

After completing this unit, you should be able to:

- Discuss the mainframe hardware and software environment
- State the purpose of an operating system
- Explain various data processing terms
- Discuss different types of programs
- Describe the need for job control language (JCL)
- Define the JOB, EXEC, and DD statements
- Differentiate between single and multistep jobs
- Explain JCL errors, return codes, and ABENDS

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597	<i>z/OS MVS JCL Reference</i>
SA22-7598	<i>z/OS MVS JCL User's Guide</i>

Unit objectives

After completing this unit, you should be able to:

- Discuss the mainframe hardware and software environment
- State the purpose of an operating system
- Explain various data processing terms
- Discuss different types of programs
- Describe the need for job control language (JCL)
- Define the JOB, EXEC, and DD statements
- Differentiate between single and multistep jobs
- Explain JCL errors, return codes, and ABENDs

© Copyright IBM Corporation 2011

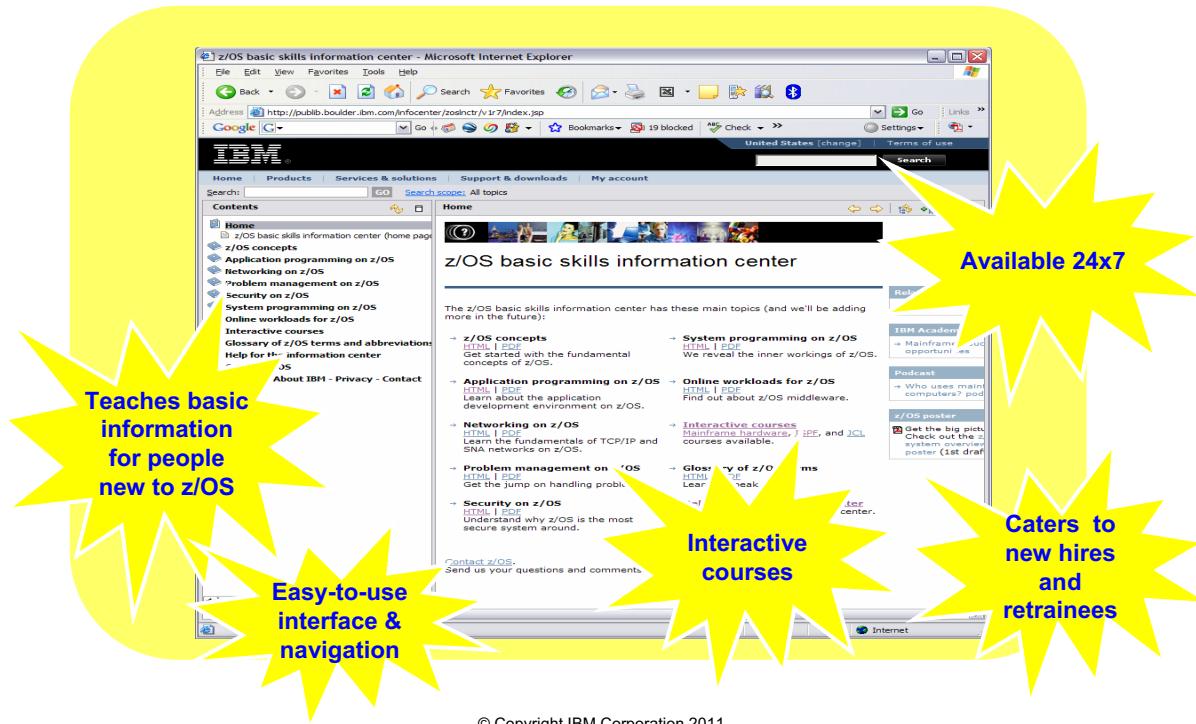
Figure 1-1. Unit objectives

ES074.0

Notes:

z/OS basic skills information center

- External website:
 - <http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp>



© Copyright IBM Corporation 2011

Figure 1-2. z/OS basic skills information center

ES074.0

Notes:

The z/OS basic skills information center provides free education for people new to z/OS.

Topics:

- Mainframe concepts
- z/OS concepts
- Application programming on z/OS
- Networking on z/OS
- Problem management on z/OS
- Security on z/OS
- z/OS system installation and maintenance
- Data and storage management on z/OS
- Online workloads for z/OS
- Database systems on z/OS
- Transaction systems on z/OS
- Web-based workloads on z/OS
- Reusable JCL collection

30-minute Interactive courses:

- Interactive System Productivity Facility (ISPF)
- JCL
- Mainframe hardware

The z/OS basic skills information center is continually adding new content.

JCL course

The screenshot shows a web browser displaying the IBM z/OS Infocenter. The address bar shows the URL: <http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp>. The page title is "Job control language (JCL) basics course". The left sidebar contains a navigation tree with categories like Home, Mainframe concepts, z/OS concepts, Application programming on z/OS, Networking on z/OS, z/OS problem management, Security on z/OS, z/OS system installation and maintenance, Data and storage management on z/OS, Online workloads for z/OS, Reusable JCL collection, 30-minute courses on z/OS (which is expanded to show "Learn in 30 minutes or less" with sub-options for Mainframe hardware basics courses, Interactive System Productivity Facility (ISPF), and Job control language (JCL) basics course), Glossary of z/OS terms and abbreviations, Notices and accessibility, and Help for the information center. The main content area displays the "30-minute courses on z/OS" section, specifically the "Job control language (JCL) basics course". It includes a brief introduction stating that the easiest way to learn JCL is to use some that's already been written, followed by a note that the course takes no more than 30 minutes to complete and requires no previous knowledge of JCL. It also mentions the "Course" and "Parent topic: Learn in 30 minutes or less". Below the main content are links for Notices, Terms of use, Support, and Contact z/OS, along with the URL: http://publib.boulder.ibm.com/infocenter/zos/basics/topic/com.ibm.zos.zcourses/zcourses_jclintro.htm.

Figure 1-3. JCL course

ES074.0

Notes:

30-minute interactive courses on mainframe hardware, ISPF, and JCL are available.

The following JCL course is available:

- *Introduction to JCL*

This first unit, JCL basics, is designed to introduce the basic concepts and structures of the z/OS job control language (JCL) so that you may quickly begin to start coding JCL statements to run jobs on z/OS. The JCL basics unit consists of two learning modules:

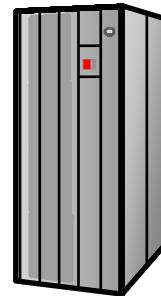
- JCL overview
- JCL syntax

These modules are designed to be completed in the order shown. Each module takes approximately 30 minutes or less to finish. After finishing these two modules, you can test your knowledge by completing the quiz questions, which are accessible through the Quiz tab.

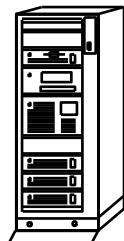
Processors

Mainframes

System z: z196, z114, z10, z9,
z990/z890, z900/z800



Midrange: System I, System p
and System x



Personal Systems: PC



© Copyright IBM Corporation 2011

Figure 1-4. Processors

ES074.0

Notes:

Many types of processors are used to process data.

- Mainframes are the most powerful. Mainframes are usually used by large companies. z/OS runs on a mainframe.
- Midrange systems are also called minicomputers. Midrange systems are often selected to fill the gap between personal systems and mainframes.
- Personal systems or personal computers are referred to as microcomputers or just PCs. They usually process work for one user at a time.

z/OS runs on the following types of processors:

- S/390 9672 in 31-bit architecture mode
- System z servers in 64-bit architecture mode:
 - z10 EC, z9 EC, z9 BC, z990, z900

Input/output devices

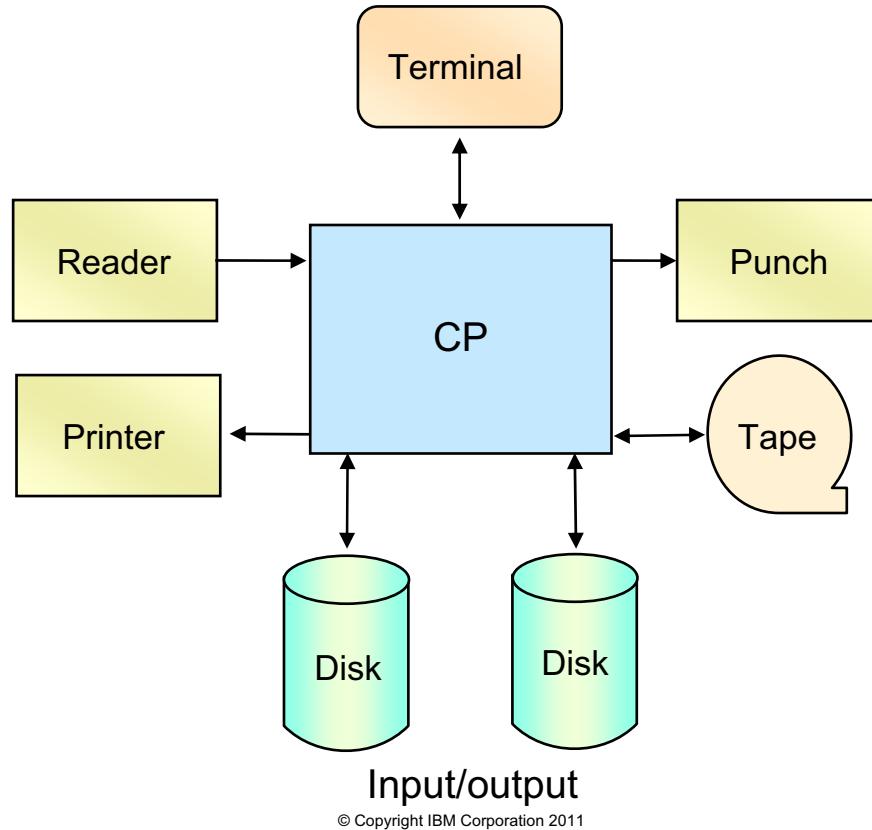


Figure 1-5. Input/output devices

ES074.0

Notes:

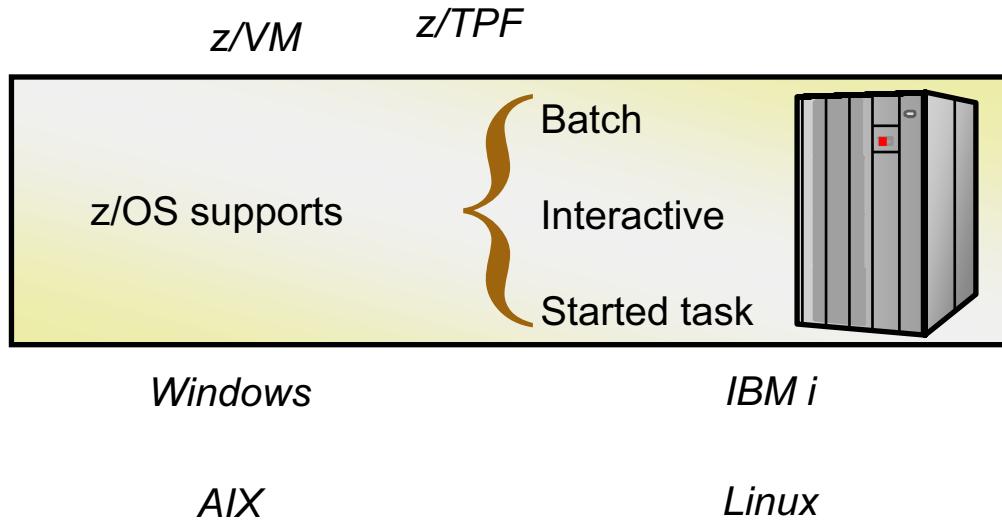
z/OS supports many different device types:

- Disk controllers (DS8800, DS8000, DS6000, ESS 2105)
- Direct access storage devices (DASDs) such as 3390s and 3380s
- Tape devices such as 3590s, 3490s, and 3480s
- Display devices (3270 device types) such as 3482s, 3192s, and 3290s
- Printers such as 4248s, 3828s, and 3800s.

Other devices may or may not be found in z/OS mainframe environments:

- Many z/OS shops no longer have hardware card readers. Punches have all but vanished from the scene.
- Miscellaneous devices such as fibre channel connection (FICON)/Enterprise Systems Connection (ESCON) directors, graphics devices, and channel-to-channel (CTC) can be found in various z/OS shops.

Operating systems



© Copyright IBM Corporation 2011

Figure 1-6. Operating systems

ES074.0

Notes:

An *operating system* is the software that controls the execution of programs and provides system services for those programs.

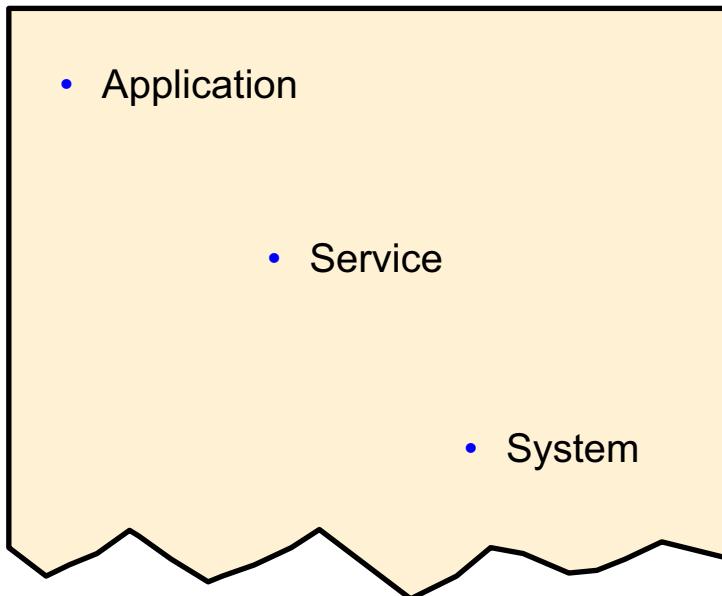
Many operating systems have been designed by multiple vendors over the years. Some of the most popular systems are listed above. This class will concentrate on z/OS. More specifically, the class will concentrate on the JCL required to request z/OS services.

z/OS is a multiprogramming, multiprocessing operating system designed to run on IBM mainframe processors.

z/OS supports features such as:

- 31-bit and 64-bit addressing
- Address spaces, data spaces, and hiperspaces
- The Storage Management Subsystem (SMS)
- ESCON/FICON architecture
- z/OS UNIX Services
- Parallel Sysplex of coupled z/OS systems

Programs



© Copyright IBM Corporation 2011

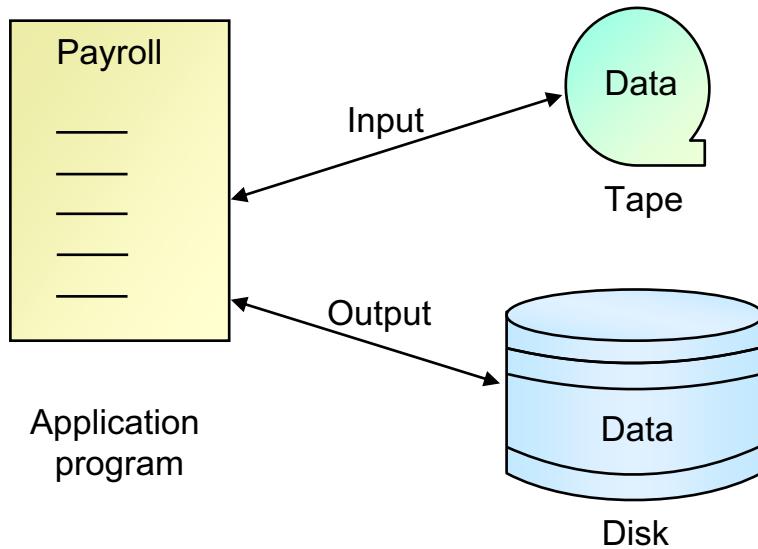
Figure 1-7. Programs

ES074.0

Notes:

- **Application:** The business programs you use; they can be written only for your location or custom tailored (for example, accounts receivable, accounts payable, and payroll).
- **Service:** Utilities, linkage editor, sorting, and compiling.
- **System:** The operating system programs that provide z/OS services support z/OS users.

Data



© Copyright IBM Corporation 2011

Figure 1-8. Data

ES074.0

Notes:

Sometimes we forget the reason for the existence of z/OS and the processor, but it is to process data. In fact, it used to be the name of our profession.

Programs usually need input and output data (payroll records, inventory records, and so forth).

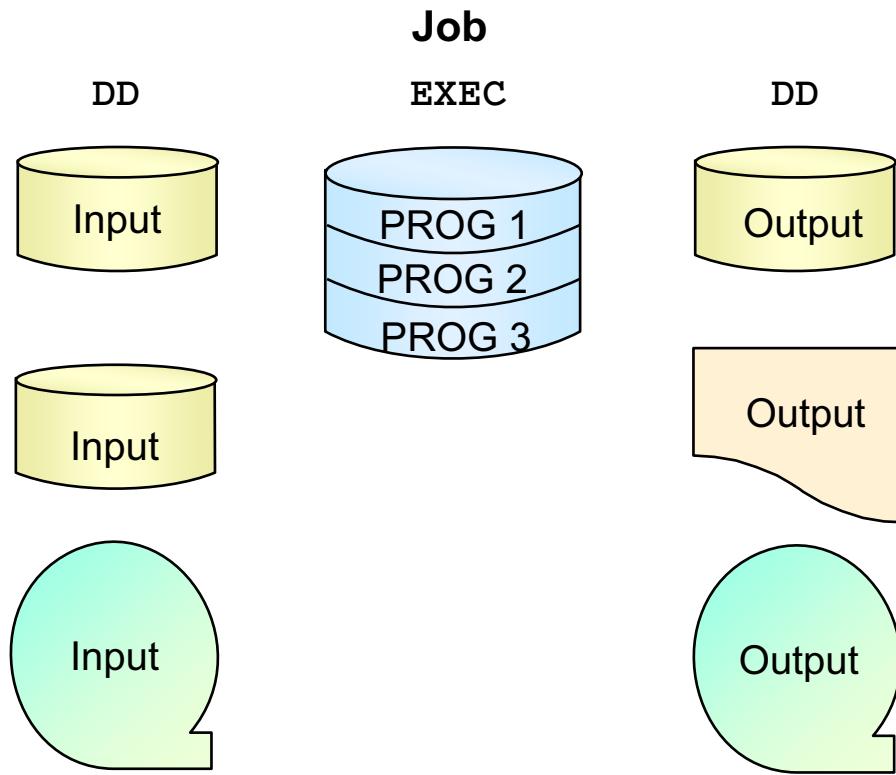
Data is stored in data sets which reside on DASDs or tapes.

Job requirements that are submitted to the system must identify these data sets.

Various data management terms are used in describing data:

Data:	Information provided to the system for processing
Field:	A specified area used for a particular category of data
Record:	A collection of related data organized in fields
Data set:	A file of related records

Job control language



© Copyright IBM Corporation 2011

Figure 1-9. Job control language

ES074.0

Notes:

For your program to execute on the computer and perform the work you designed it to do, your program must be processed by your operating system.

z/OS operating system consists of an MVS/SP base control program (BCP) with a job entry subsystem (JES2 or JES3) and Data Facility Storage Management Subsystem/Multiple Virtual Storage (DFSMS/MVS) DFSMSdfp installed with it.

For the operating system to process a program, programmers must perform certain job control tasks. These tasks are performed through the job control statements, which consist of:

- JCL statements
- JES2 control statements
- JES3 control statements

Job control language (JCL) is used to tell the system what program to execute, followed by a description of program inputs and outputs. It is possible to submit JCL for batch processing or start a JCL procedure (PROC), which is considered a started task. The

details of JCL can be complicated, but the general concepts are quite simple. Also, a small subset of JCL accounts for at least 90% of what is actually used.

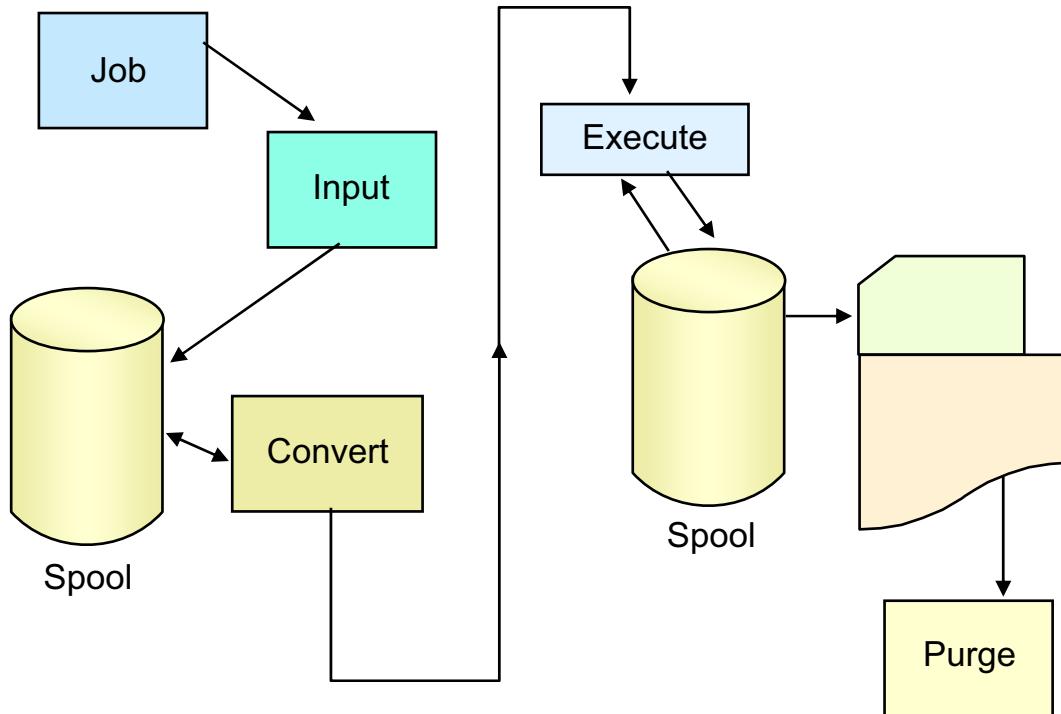
JCL describes job requirements to the system. For example, JCL specifies information about:

- Programs
- Data sets
- Devices
- Volumes
- Space
- Data set attributes

JCL statements are used to input this information to the system.

-

JES responsibilities



© Copyright IBM Corporation 2011

Figure 1-10. JES responsibilities

ES074.0

Notes:

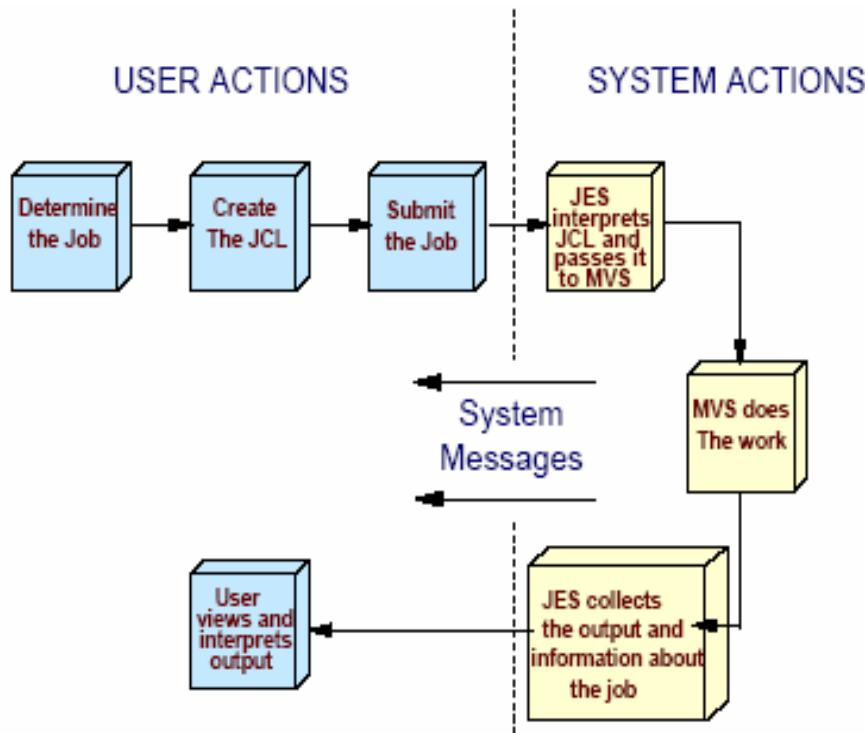
z/OS shares the management of work with a Job Entry Subsystem (JES) which manages work before and after execution. Although there are two job entry subsystems, JES2 and JES3, both perform the same general functions:

- JES accepts work
- Prepares work for execution
- Temporarily stores work on DASD until z/OS is ready to accept it
- Selects jobs for z/OS execution

As z/OS executes the work, JES:

- Handles printed output
- Purges it from the system when the work completes

JCL-related actions



© Copyright IBM Corporation 2011

Figure 1-11. JCL-related actions

ES074.0

Notes:

The visual shows an overview of the job submission process. The user performs the activities on the left side of the figure, and the system performs those on the right. In this example, z/OS and JES comprise the “system”.

For every job that you submit, you need to tell the system where to find the appropriate input, how to process that input (that is, what program or programs to run), and what to do with the resulting output.

You use JCL to convey this information to the system through a set of statements known as *job control statements*. JCL’s set of job control statements is quite large, enabling you to provide a great deal of information to the system.

Most jobs, however, can be run using a very small subset of these control statements. Once you become familiar with the characteristics of the jobs you typically run, you may find that you need to know the details of only some of the control statements.

Within each job, the control statements are grouped into job steps.

JCL statements

JOB	NULL	COMMAND
EXEC	DELIMITER	IF/THEN/ELSE/ENDIF
DD	OUTPUT	INCLUDE
PROC	JCL COMMAND**	JCLLIB
PEND	CNTL	SET
COMMENT	ENDCNTL	XMIT***

© Copyright IBM Corporation 2011

Figure 1-12. JCL statements

ES074.0

Notes:

The JOB, EXEC, and DD statements have many parameters to allow the user to specify instructions and information. Describing them all would fill an entire book (such as the IBM publication *z/OS JCL Reference*).

This section provides only a brief description of a few of the more commonly used parameters for the JOB, EXEC, and DD statements.

- There are 18 JCL statements.
- The three most important JCL statements in most environments are:

- JOB
- EXEC
- DD

Note:

- ** Supported only on JES2 systems
- *** Supported only on JES3 systems

JOB statement

- Defines a job and job-related information to the system
 - Accounting information
 - Programmer
 - Class
 - Storage required
 - Conditional testing

Example:

```
//MYJOB JOB 1,NOTIFY=&SYSUID,REGION=6M
```

© Copyright IBM Corporation 2011

Figure 1-13. JOB statement

ES074.0

Notes:

The JOB statement must be the first JCL statement in each job. There is exactly one JOB statement in each job.

The JOB statement marks the beginning of a job and specifies processing information.

The JOB statement //MYJOB JOB 1 has a job name: MYJOB. The 1 is an accounting field that can be subject to system exits that might be used for charging system users.

Some common JOB statement parameters include:

- REGION: Requests specific memory resources to be allocated to the job.
- NOTIFY: Sends notification of job completion to a particular user, such as the submitter of the job.
- USER: Specifies that the job is to assume the authority of the user ID specified.
- TYPRUN: Delays or holds the job from running, to be released later.
- CLASS: Directs a JCL statement to execute on a particular input queue.
- MSGCLASS: Directs job output to a particular output queue.
- MSGLEVEL: Controls the number of system messages to be received.

EXECUTE statement

- Defines a job step and job step-related information to the system
 - What program (or procedure) to run
 - Conditional testing
 - Parameters to be passed to program

Example:

/ /MYSTEP EXEC PGM=SORT

© Copyright IBM Corporation 2011

Figure 1-14. EXECUTE statement

ES074.0

Notes:

The EXEC statement marks the beginning of a step in the job.

The EXEC statement must be the first JCL statement in each step.

The EXEC statement identifies the program or procedure to be executed and specifies how to process the step.

A job can have a maximum of 255 job steps.

The EXEC JCL statement //MYSTEP EXEC has a step name of MYSTEP. Following the EXEC is either PGM= (executable program name) or a JCL procedure name. When a JCL PROC is present, the parameters will be the variable substitutions required by the JCL PROC.

Common parameters found on the EXEC PGM= statement are:

- PARM: Parameters known by and passed to the program.
- COND: Boolean logic for controlling execution of other EXEC steps in this job.
- IF, THEN, ELSE: JCL statements exist that are superior to using COND; however, lots of old JCL might exist in production environments using this statement.
- TIME: Imposes a time limit.

Data definition statement

- Defines data requirements for the program
 - Describes a data set
 - Specified input and output resources for data set

© Copyright IBM Corporation 2011

Figure 1-15. Data definition statement

ES074.0

Notes:

DD statements describe the input and output data sets and their characteristics.

DD statements normally follow an EXEC statement, but there are exceptions. The JOBLIB DD statement precedes all EXEC statements. DD statements can usually be arranged in any order following an EXEC statement.

DD statements are sometimes used to route print/punch data sets to JES for output processing.

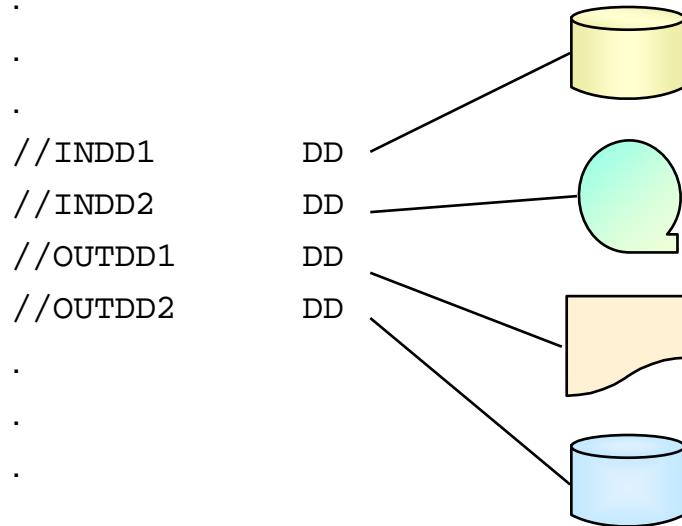
The DD JCL statement //MYDATA DD has a ddname of MYDATA. The DD statement has significantly more parameters than the JOB or EXEC statements. The DD JCL statement can be involved with many aspects of defining or describing attributes of the program inputs or outputs.

Some common DD statement parameters (covered in more detail in next unit) are:

- DSN
- DISP
- SPACE
- SYSOUT
- VOL=SER
- UNIT
- DEST
- DCB

Input/output data

Job stream



© Copyright IBM Corporation 2011

Figure 1-16. Input/output data

ES074.0

Notes:

The example job step has multiple inputs:

- INDD1 input from disk data set
- INDD2 input from tape data set

And multiple outputs:

- OUTDD1 output to the printer
- OUTDD2 output to disk data set

Example:

```

//JOB1    JOB    1234,SMITH
//STEP1   EXEC   PGM=PAYROLL
//INDD1   DD     DSN=TIMECARD.DALLAS,DISP=OLD
//INDD2   DD     DSN=TIMECARD.AUSTIN,DISP=OLD
//OUTDD1  DD     SYSOUT=*
//OUTDD2  DD     DSN=APRIL.PAYROLL,
//          DISP=(,CATLG), .....
  
```

Multistep job

```
//JOB1      JOB  
//STP1      EXEC  
//DD1       DD  
//DD2       DD  
//STP2      EXEC  
//DD1       DD  
//DD2       DD  
//STP3      EXEC  
//DD1       DD  
//DD2       DD  
//STP4      EXEC  
//DD1       DD  
//DD2       DD
```

© Copyright IBM Corporation 2011

Figure 1-17. Multistep job

ES074.0

Notes:

A job step consists of all the control statements needed to run one program, for example, a sort, a copy, or an application program. If a job needs to run more than one program, the job will contain a different job step for each of those programs. A job can have from one up to 255 steps.

- The above example shows a job consisting of four steps.
- A one-step job would contain only one EXEC statement.
- A job can have 1 to 255 steps. Can you envision a situation where you might want to create a job with more than 255 steps?

JCL errors

```

//JOB1           JOB
//STP1           EXEC
//DD1            DD
//DD2            DD
//STP2           EXEC
JCL error
//DD1            DD
//DD2            DD
//STP3           EXEC
//DD1            DD
//DD2            DD
//DD3            DD

```

Bypass these steps

© Copyright IBM Corporation 2011

Figure 1-18. JCL errors

ES074.0

Notes:

There are two types of JCL errors:

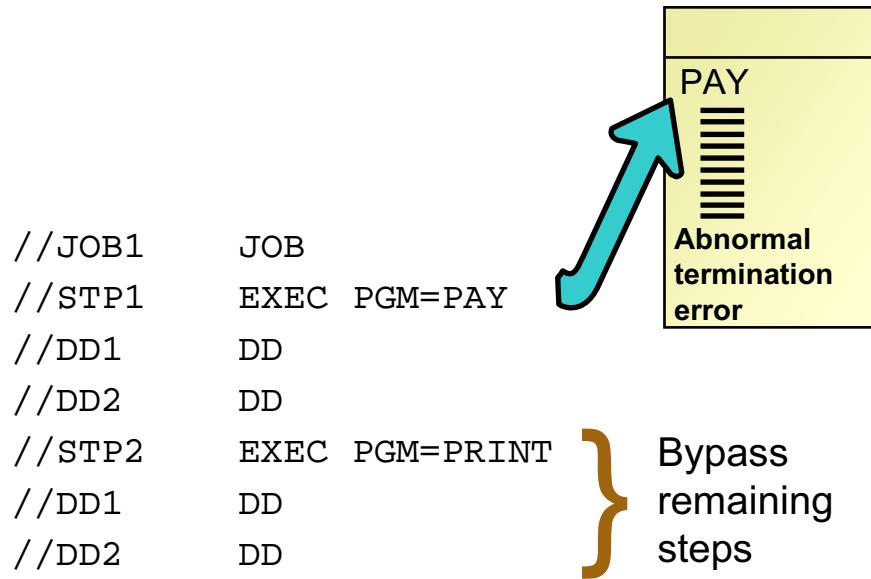
- JCL syntax errors
- JCL execution time errors

If a JCL syntax error is detected, the system bypasses the entire job and does not attempt job execution.

If a JCL execution time error is detected, the system bypasses all remaining steps, including the step containing the error. Misspelling a data set name can produce a JCL execution time error.

The above example illustrates a JCL execution time error.

Program abnormal termination



© Copyright IBM Corporation 2011

Figure 1-19. Program abnormal termination

ES074.0

Notes:

If an executing program abnormally ends (ABENDs), by default all remaining job steps are bypassed; however, JCL allows this default action to be overridden.

Messages indicate when an ABEND has occurred.

Programmers can write recovery routines that sometimes enable the system to recover from errors. Thus, some ABENDs can be avoided.

Return codes



© Copyright IBM Corporation 2011

Figure 1-20. Return codes

ES074.0

Notes:

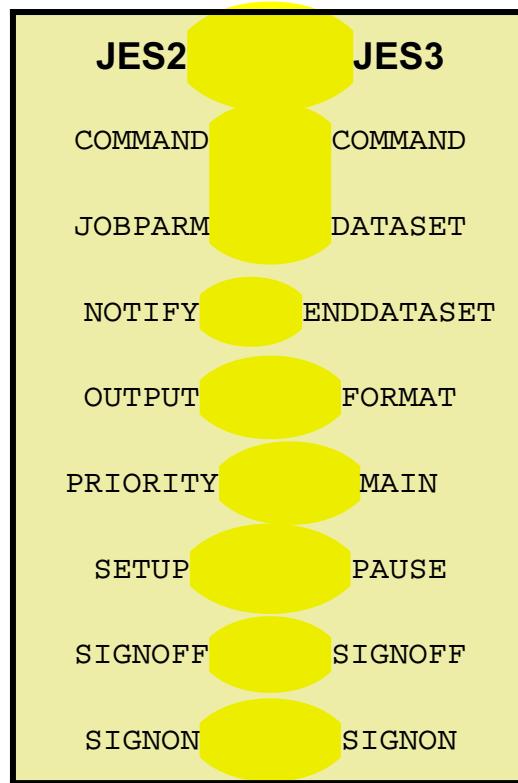
Programs that do not ABEND assign a return code to signify a certain condition.

Return codes can be any whole number from 0 to 4095.

Most IBM-developed programs produce standard return codes of 0, 4, 8, 12, or 16.

A return code of 0 usually indicates successful execution. A high return code usually indicates that a step failed.

JES2/JES3 control statements



© Copyright IBM Corporation 2011

Figure 1-21. JES2/JES3 control statements

ES074.0

Notes:

JES2/JES3 control statements are used to control the job and supply parameters to the job.

Most JES2 statements begin with /* in columns one and two.

Most JES3 statements begin with ///* in columns one, two, and three.

Checkpoint (1 of 2)

1. Which of the following are valid Job Entry Subsystems?
 - a. JES1
 - b. JES2
 - c. JES3
2. What is the JCL statement to define a job step and job step-related information to the system?
3. True or False: A job can have a maximum of 256 job steps.
4. True or False: If a JCL syntax error is detected, the system bypasses the entire job.
5. True or False: If a JCL execution time error is detected, the system bypasses the entire job.

© Copyright IBM Corporation 2011

Figure 1-22. Checkpoint (1 of 2)

ES074.0

Notes:

Checkpoint (2 of 2)

6. List two ways to submit JCL work in the system.
 - a.
 - b.
7. Name three basic JCL statements.
 - a.
 - b.
 - c.
8. Valid return codes are in which of the following ranges:
 - a. 0 – 4095
 - b. 0 – 4096
 - c. 0 – 8192
9. True or False: JES2 and JES3 support the same syntax language.

© Copyright IBM Corporation 2011

Figure 1-23. Checkpoint (2 of 2)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Discuss the mainframe hardware and software environment
- State the purpose of an operating system
- Explain various data processing terms
- Discuss different types of programs
- Describe the need for job control language (JCL)
- Define the JOB, EXEC, and DD statements
- Differentiate between single and multistep jobs
- Explain JCL errors, return codes, and ABENDs

© Copyright IBM Corporation 2011

Figure 1-24. Unit summary

ES074.0

Notes:

Unit 2. JOB, EXEC, and DD statements

What this unit is about

The three most important JCL statements are the JOB, EXEC, and DD statements. Many jobs contain only these three JCL statements. This unit discusses the purpose of these statements and looks at the use of parameters on these statements. JCL statement syntax and coding rules are also discussed.

What you should be able to do

After completing this unit, you should be able to:

- Describe the JCL statement format
- Code JOB statements and JOB statement parameters
- Code EXEC statements to invoke a program and pass PARM parameters to the program
- Introduce the DD statement and a few of its parameters
- Discuss the COMMENT statement

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597

z/OS MVS JCL Reference

SA22-7598

z/OS MVS JCL User's Guide

Unit objectives

After completing this unit, you should be able to:

- Describe the JCL statement format
- Code JOB statements and JOB statement parameters
- Code EXEC statements to invoke a program and pass PARM parameters to the program
- Introduce the DD statement and a few of its parameters
- Discuss the COMMENT statement

© Copyright IBM Corporation 2011

Figure 2-1. Unit objectives

ES074.0

Notes:

JCL statement format

CCCC	C C
OOOO	O O
LLLL	L L
1234	7172

```
//NAME OPERATION PARAMETER,  
// PARAMETER, PARAMETER
```



4 - 16

© Copyright IBM Corporation 2011

Figure 2-2. JCL statement format

ES074.0

Notes:

A JCL statement consists of one or more 80-byte records.

First of all, you need to know the basic JCL codification rules.

- The JCL statements start in column 1 and are identified by // at the beginning of the line.
- The commentary lines are identified by ///* at the start of the line.
- Statements are coded from column 1 to column 71.
- A comma indicates that the statement has continuation.
- A continuation of a statement must start between columns 4 and 16.
- // and the rest of statement with blanks indicates the end of the job.
- Keywords are in uppercase.
- Comments must be separated from the operation parameter by at least one blank.

Refer to *z/OS MVS JCL Reference* for continuation and other coding rules.

Each statement is divided into the following five fields:

IDENTIFIER field

- Indicates a JCL statement:
 - // in columns 1 and 2 of all JCL statements except the delimiter
 - /* or installation designated characters in columns 1 and 2 as a delimiter
 - ///* in columns 1, 2, and 3 to mark a comment statement

NAME field

- Identifies a statement so that it can be referred to later:
 - Must begin in column 3
 - One to eight characters in length (alphanumeric or national (#, @, \$))
 - First character must be alphabetic or national
 - Must be followed by at least one blank

OPERATION field

- Specifies the statement or command:
 - Follows the **NAME** field
 - Must be preceded and followed by at least one blank
 - Consists of characters specified in syntax for statement

PARAMETER field

- Contains parameters separated by commas:
 - Follows the **OPERATION** field
 - Must be preceded and followed by at least one blank
 - Consists of two types:
 - Positional
 - Keyword

COMMENT field

- Can contain any information:
 - Follows the **PARAMETER** field
 - Must be preceded by at least one blank
 - Difficult to use. **COMMENT** statement is recommended.

Parameters

```
//NAME OPERATION P1,P2,P3,K1=A  
//          K2=B,K3=(P1,K1=T)
```

© Copyright IBM Corporation 2011

Figure 2-3. Parameters

ES074.0

Notes:

A JCL statement has two kinds of parameters: positional and keyword. All parameters are optional.

- **Positional:** If needed, must be coded first in the order described in the *z/OS MVS JCL Reference manual*.
 - If a leading positional parameter is omitted and trailing positional parameters follow, code a comma to indicate its absence.
 - If the last positional parameter is omitted, no comma is necessary.
 - If all positional parameters are omitted, no commas are necessary.
- **Keyword:** Code keyword parameters, in any order, after required positional parameters.
 - Keyword parameters are indicated by coding the keyword followed by an '=' sign and the keyword value.
 - Commas are not coded to indicate the omission of keyword parameters.

Parameter examples

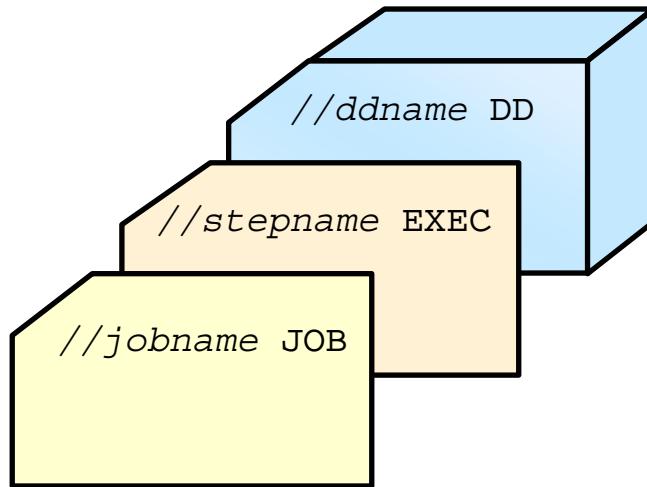
```
//DD1    OPER    POS1, POS2, KEY3=C, KEY1=A  
//DD2    OPER    , POS2, KEY2=B  
//DD3    OPER    POS1, KEY2=B  
//DD4    OPER    KEY3=C, KEY1=A
```

- **Subparameter lists:** Parameters can contain a list of items called a *subparameter list*.
 - Can contain both positional and keyword subparameters.
 - Must be enclosed in parentheses or apostrophes unless the list contains a single keyword subparameter.

Subparameter examples

```
//DD1    OPER    POS1, KEY4=(P1, K2=T, K1=Q)  
//DD2    OPER    , , POS3, KEY4=K1=Q
```

The JOB statement



© Copyright IBM Corporation 2011

Figure 2-4. The JOB statement

ES074.0

Notes:

A JOB statement marks the beginning of a job and assigns a name to the job. The JOB statement is also used to provide certain administrative information, including security, accounting, and identification information.

Every job has one and only one JOB statement.

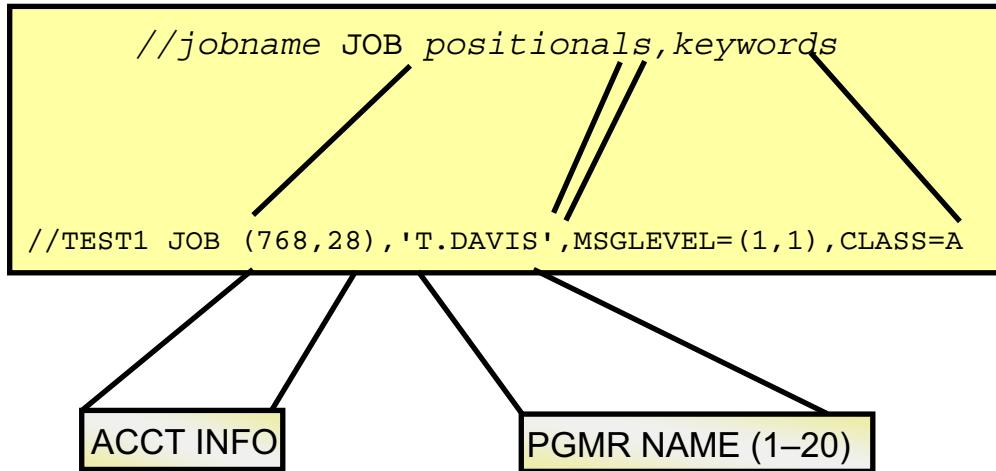
The JOB statement must be:

- The *first* statement in the job
- The *only* JOB statement in the job

The job name (required):

- Must begin in column 3
- Is one to eight characters in length (alphanumeric or national (#, @, \$))
- Must have a first character that is alphabetic or national
- Must be followed by at least one blank

JOB statement syntax



© Copyright IBM Corporation 2011

Figure 2-5. JOB statement syntax

ES074.0

Notes:

The JOB statement syntax has the following parts:

- // in columns 1 and 2.
- Job name (required).
- JOB in the operation field.
- Positional parameters:

- Accounting information:

- A maximum of 143 characters can be coded between the enclosing parentheses or apostrophes.
- If special characters are used, either the subparameter or the entire accounting information parameter must be enclosed in apostrophes.

- Programmer name:

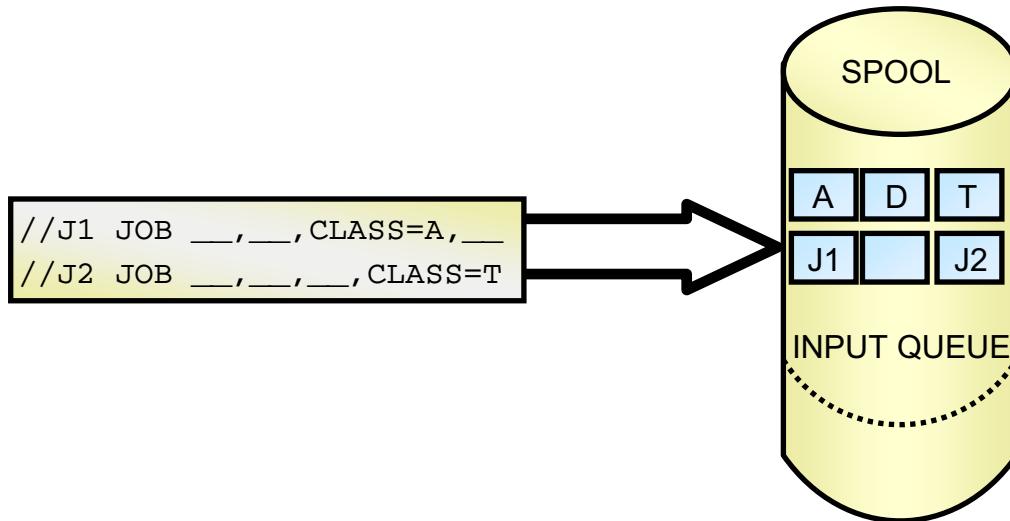
- A maximum of 20 characters can be coded, including special characters. If special characters (other than hyphens, leading periods, or embedded periods) are coded, the name must be enclosed in apostrophes.

The following keyword parameters, a subset of about 30 available, will be covered in this course:

CLASS	Assigns the job to a jobclass.
COND	Specifies rules for conditional job execution.
LINES	Limits print output before system takes action.
MSGCLASS	Assigns the job log to an output class.
MSGLEVEL	Shows information to be placed in the job log.
NOTIFY	Notifies the TSO user when a job is complete.
REGION	Specifies maximum virtual storage size for a job.
MEMLIMIT	Controls the amount of usable virtual storage above 2 GB.
RESTART	Controls the restart of a job.
TIME	Specifies the maximum time a job can execute.
TYPRUN	Requests special job processing.

All parameters are optional; however, your installation might require the accounting information parameter and the programmer's name parameter.

CLASS parameter



© Copyright IBM Corporation 2011

Figure 2-6. CLASS parameter

ES074.0

Notes:

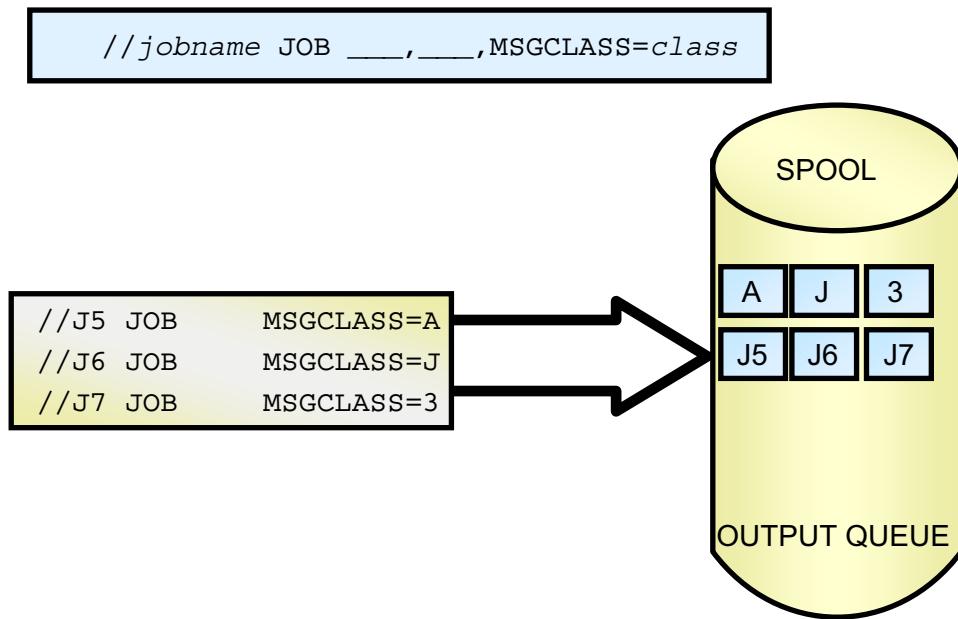
Job class

The CLASS= field identifies the JES job class this job will execute under.

In the example, CLASS=A is the JES job class. Many sites do not use this option, and the JES class is set according to your user ID. Job classes are set up at JES initialization.

- The class places your job into a JES input queue class.
- CLASS is one character, A to Z or 0 to 9.
- If you do not specify a class, JES uses the installation default specified at initialization.

MSGCLASS parameter



© Copyright IBM Corporation 2011

Figure 2-7. MSGCLASS parameter

ES074.0

Notes:

MSGCLASS= class

MSGCLASS= assigns the job log to an output class. The output class and its characteristics are identified in a parameter file used at JES initialization.

- MSGCLASS controls the destination of the job log.
- MSGCLASS is one character, A to Z or 0 to 9.
- The default is based on the source of the job.

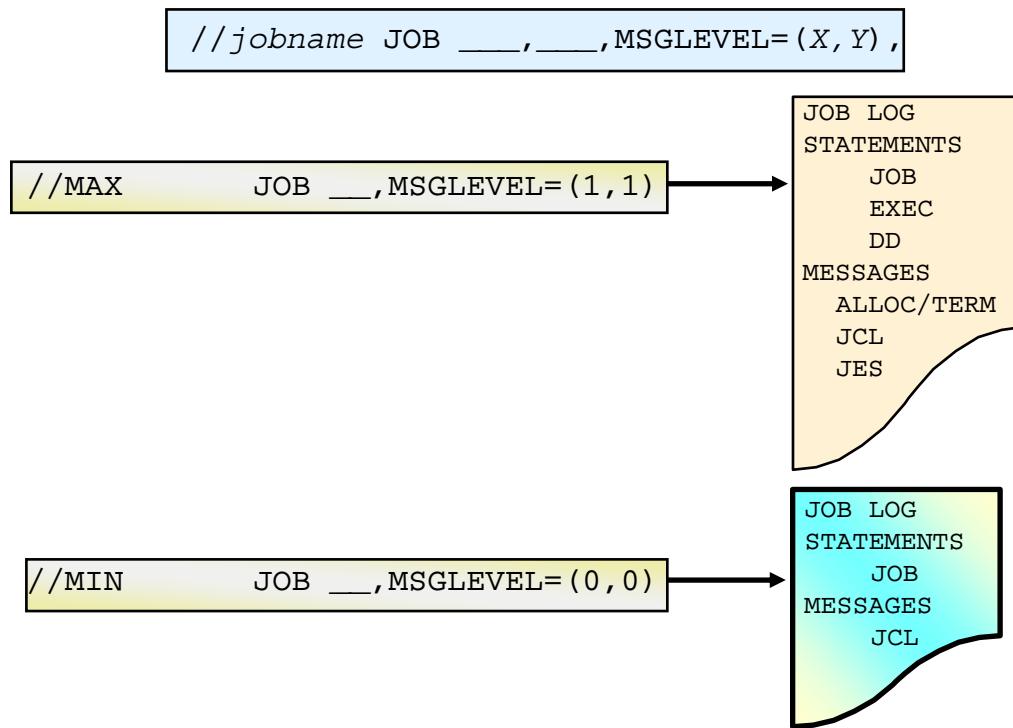


Note

The following two output classes are commonly defined:

- **A:** Print on standard forms
- **B:** Punch

MSGLEVEL parameter



© Copyright IBM Corporation 2011

Figure 2-8. MSGLEVEL parameter

ES074.0

Notes:

The **first subparameter** of MSGLEVEL controls which statements will be printed in the job log.

- **MSGLEVEL=0:** Print only the JOB statement
- **MSGLEVEL=1:** Print all JCL and JES statements, including all statements in procedures.
- **MSGLEVEL=2:** Print only submitted JCL and JES statements. Statements in procedures are not printed.

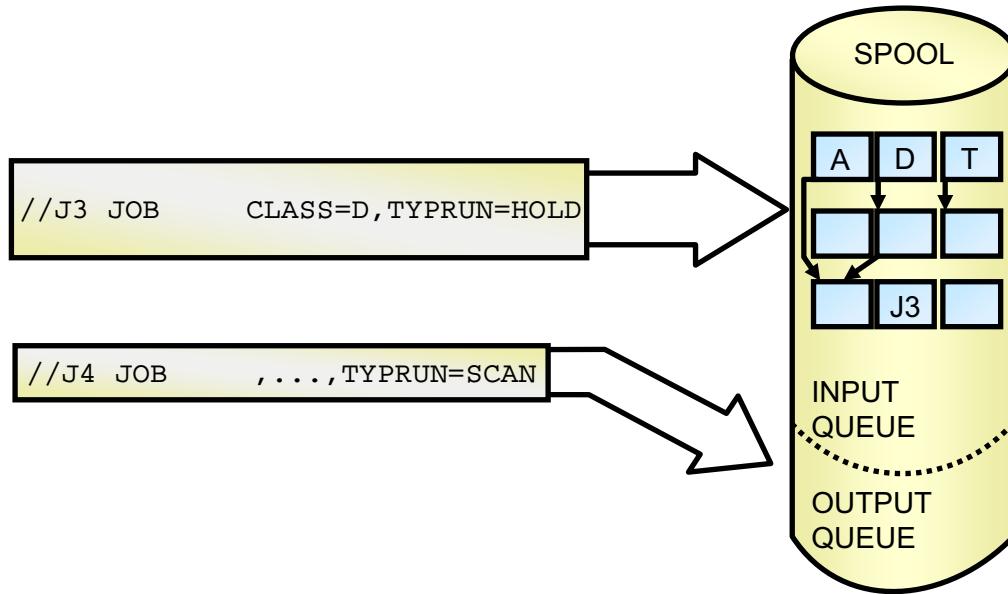
The **second subparameter** of MSGLEVEL controls which messages will be printed in the job log.

- **MSGLEVEL=(, 0) :** If normal termination, print only JCL messages. If abnormal termination, print all messages.
- **MSGLEVEL=(, 1) :** All messages are printed regardless of how the job terminates.

Defaults: If MSGLEVEL is omitted, JES supplies an installation default when JES is initialized.

NOTIFY and TYPRUN parameters

//jobname JOB ____,____,NOTIFY=tsoid,TYPRUN=option



© Copyright IBM Corporation 2011

Figure 2-9. NOTIFY and TYPRUN parameters

ES074.0

Notes:

NOTIFY tells the system where to send job complete information.

- The NOTIFY parameter causes the system to notify the TSO user specified on the NOTIFY parameter when the job completes.
 - Code NOTIFY=USERID. *UserID* must be a valid TSO user ID.
 - &SYSUID tells the system to automatically insert your user ID here, so that the information will be sent to you.
- The TYPRUN parameter modifies the way JES processes your job:

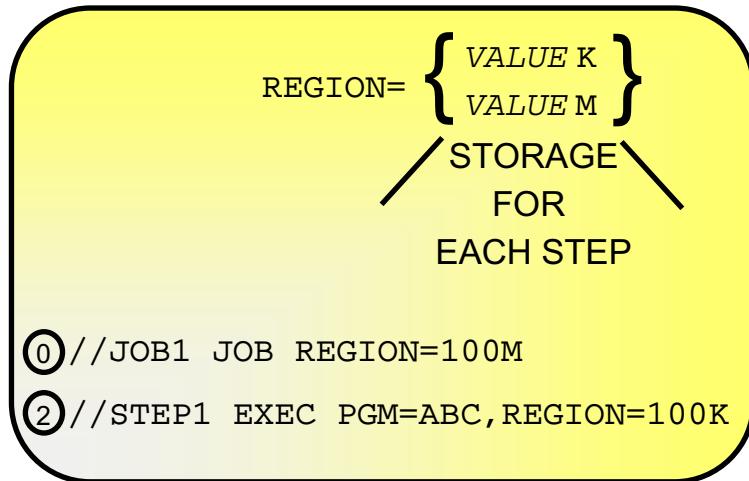
TYPRUN=SCAN The JCL is scanned for syntax errors but is not executed.

TYPRUN=HOLD The job is held in the input queue. The operator must release the job to execute it.

TYPRUN=COPY The JCL is copied as submitted to the sysout class specified in the MSGCLASS parameter (JES2 ONLY).

TYPRUN=JCLHOLD Requests that JES2 hold the job before completing JCL processing. JES2 holds the job until the operator releases it.(JES2 ONLY).

REGION parameter



© Copyright IBM Corporation 2011

Figure 2-10. REGION parameter

ES074.0

Notes:

The REGION parameter specifies the amount of virtual storage needed by a job or job step. It can be coded on the JOB or EXEC statement.

- When REGION is specified on the JOB statement, it applies to each step in the job and overrides the REGION parameter on each EXEC statement.
- A job will ABEND if some step needs a larger REGION size or if the REGION value cannot be obtained.
- When REGION is specified on the EXEC statement, it specifies the amount of space required by the step. It should be used when different steps require different amounts of space.
- A step will ABEND if a step needs a larger REGION or if the REGION value cannot be obtained.

REGION can be specified with the value equal to K (kilobytes) or M (megabytes).

A value equal to 0K or 0M gives the job all the storage available.

REGION example on the EXEC statement:

```
//INLINE JOB (ACCT#),'IN-LINE JCL',  
//      CLASS=A,  
//      MSGCLASS=X,  
//      MSGLEVEL=(1,1),  
//      REGION=6M  
  
//STEP1    EXEC PGM=ABLE,REGION=256K
```

MEMLIMIT parameter

```
//TC1 JOB MEMLIMIT=50G,REGION=0M  
  
//TC2 JOB MEMLIMIT=125M,TIME=NOLIMIT  
  
//TC3 JOB MEMLIMIT= 9T,MSGLEVEL=1  
  
//TC4 JOB REGION=3M,MEMLIMIT=16384P  
  
//TC5 JOB REGION=125M,MSGLEVEL=(1,1),  
MEMLIMIT=NOLIMIT,MSGCLASS=A
```

© Copyright IBM Corporation 2011

Figure 2-11. MEMLIMIT parameter

ES074.0

Notes:

Use the **MEMLIMIT** parameter to specify the limit on the total number of usable virtual pages above the bar (2 GB) for a single address space.

MEMLIMIT can be specified with the value equal to M (megabytes), G (gigabytes), T (terabytes), or Petabytes).

If no **MEMLIMIT** parameter is specified, the default is the value defined to SMF, except when **REGION=0K/0M** is specified, in which case the default is **NOLIMIT**.

NOLIMIT specifies that there is no limit on the virtual pages to be used above the bar. Unlike the **REGION** parameter, **MEMLIMIT=0M** (or equivalent in G, T, or P) means that the step can not use virtual storage above the bar.

LINES parameter

//jobname JOB __,___,LINES=(nnnnnn,action)

//MAXLINE1 JOB LINES=(50,CANCEL)

//MAXLINE2 JOB LINES=(50,DUMP)

//MAXLINE3 JOB LINES=(50,WARNING)

© Copyright IBM Corporation 2011

Figure 2-12. LINES parameter

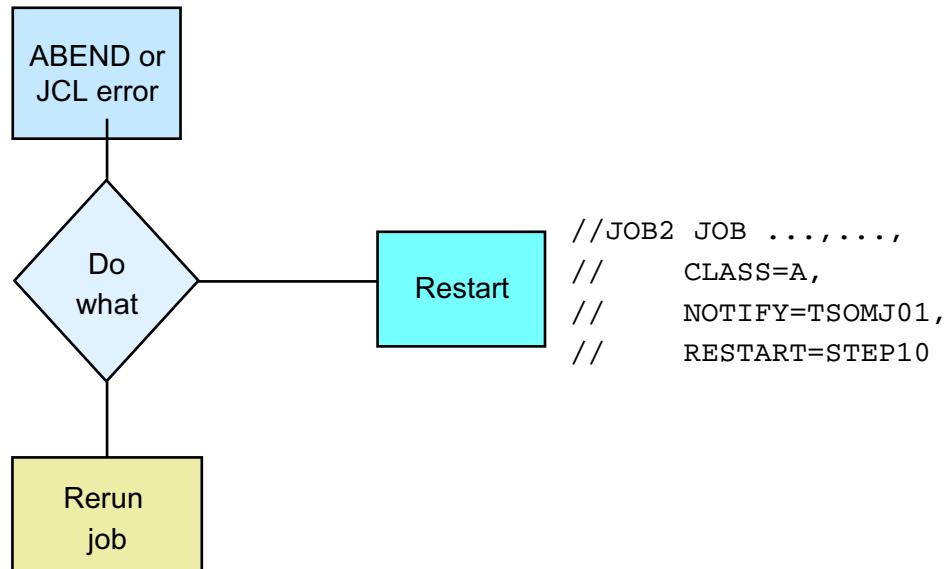
ES074.0

Notes:

The LINES parameter is used to limit the amount of output to be printed for a job's SYSOUT data sets. The system can take any of the following actions if this maximum is exceeded:

//MAXLINE1	JOB LINES=(50,CANCEL)	(If print output exceeds 50,000 lines, cancel the job.)
//MAXLINE2	JOB LINES=(50,DUMP)	(If print output exceeds 50,000 lines, take a dump.)
//MAXLINE3	JOB LINES=(50,WARNING)	(If print output exceeds 50,000 lines, issue message to operator.)

RESTART parameter



© Copyright IBM Corporation 2011

Figure 2-13. RESTART parameter

ES074.0

Notes:

Errors are a fact of life in data processing. By planning ahead, you can ask the system to restart your job after a failure (in the event that a failure does indeed occur). Of course, the restart might or might not be successful. It depends on what type of error you encountered.

Four restart options are available:

- **Automatic step restart:** The system attempts to automatically restart from the beginning of the failing step.
- **Automatic checkpoint restart:** The system attempts to automatically restart from some point within the failing step.
- **Deferred step restart:** The system allows you to examine the failure. You can make changes and then resubmit the job. The system attempts to restart from the beginning of the failing step.
- **Deferred checkpoint restart:** The system allows you to examine the failure. You can make changes and then resubmit the job. The system attempts to restart from some point within the failing step.

Use the RESTART parameter for deferred restarts to indicate the step where system is to restart a job.

- The job stream in the visual illustrates a job that is being resubmitted after an error. The submitter is requesting a deferred step restart.
- To use the RESTART option, step names must be unique.

RESTART example:

```
//JOBNAME    JOB    ACCOUNTING, 'PRGMR NAME', CLASS=A,  
//          NOTIFY=TSOMJ01,RESTART=STEP10
```

JOB statement examples

```
//TEST2    JOB (DEPT378,399216),LEON,CLASS=T  
  
//$UPDATE JOB DEPT378,TOM,CLASS=M,MSGLEVEL=(1,1)  
  
//SIMPLE JOB  
  
//SYSTEM JOB ,SYSTEM,CLASS=S,MSGLEVEL=(0,0)  
  
//SUBMIT JOB MACHE999999,'R.J. Y',NOTIFY=TSOID9,CLASS=A,REGION=512K
```

© Copyright IBM Corporation 2011

Figure 2-14. JOB statement examples

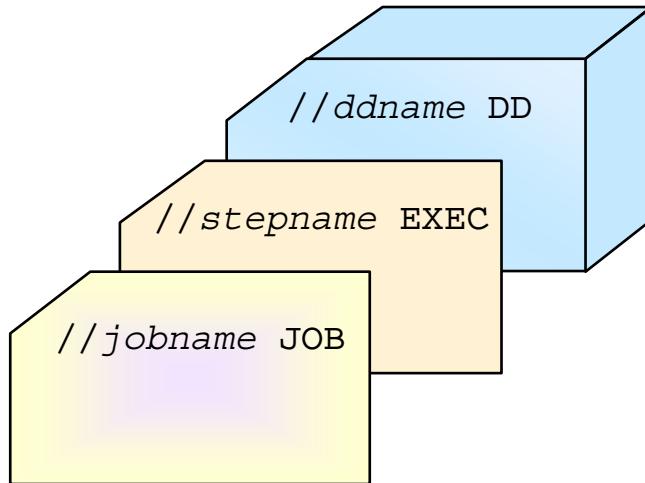
ES074.0

Notes:

Account: Some sites use this field for accounting and job processing information. In the second example, the value is DEPT378.

Programmer name: The next field is the programmer name field, which you can use to identify the member name, for example. In the second example, TOM is the programmer's name.

The EXEC statement



© Copyright IBM Corporation 2011

Figure 2-15. The EXEC statement

ES074.0

Notes:

The EXEC statement identifies the step and the program to be executed.

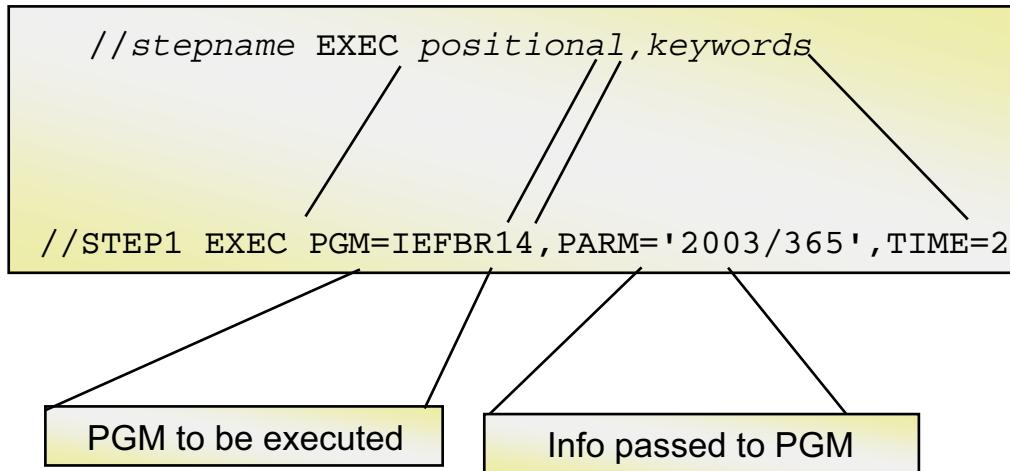
- The EXEC statement defines the beginning of a step in a job or a procedure.
- An EXEC statement is required for each job step.
- There is a maximum of 255 steps (EXEC statements) in a single job.
- Step names follow the same rules for all names in JCL:
 - One to eight characters in length (alphanumeric or national (#, @, \$)).
 - First character must be alphabetic or national.
 - Must be followed by at least one blank.



Important

It is a good practice to use unique step names within a job.

EXEC statement syntax



© Copyright IBM Corporation 2011

Figure 2-16. EXEC statement syntax

ES074.0

Notes:

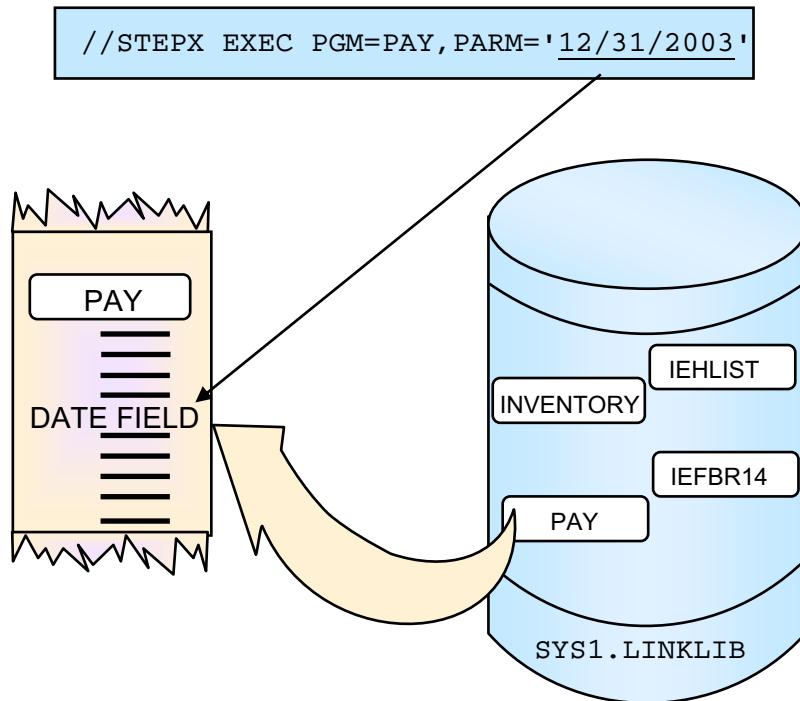
The EXEC statement syntax has the following parts:

- // in columns 1 and 2.
- Step name (not required, but recommended).
- Operation of EXEC.
- Positional parameter:
 - PGM= or PROC= is a *positional parameter* even though it is coded in keyword format.

The following lists a most used subset of EXEC statement keyword parameters:

ACCT	Allows job steps to be charged to different account codes.
COND	Specifies rules for conditional step execution.
PARM	Passes parameters to the program.
REGION	Specifies the virtual storage size for a step.
TIME	Specifies the maximum time the step is allowed to execute.

Program execution



© Copyright IBM Corporation 2011

Figure 2-17. Program execution

ES074.0

Notes:

By default, the SYSTEM LINKLIST (including SYS1.LINKLIB) is searched for the program to be executed.

The LINKLIST is defined in a linklist set in member PROGxx of the PARMLIB concatenation.

Private user libraries can be searched before SYS1.LINKLIB by using:

- A JOBLIB DD statement
- A STEPLIB DD statement

PARM continuation and limit

```
//FORMAT1      EXEC PGM=IOEZADM,
-----1-----2-----3-----4-----5-----6-----7-----
// PARM=( 'create -filesystem OMVS.PRICHAR.FS3 -aggregate
          OMVS.PRICHAR.Mx
//                      ULTI.AGGR002 -size 2000')
//SYSPRINT   DD SYSOUT=*
//STDOUT     DD SYSOUT=*
//STDERR     DD SYSOUT=*
//CEEDUMP    DD SYSOUT=*
```

```
//PARMTEST JOB (0000),TESTER,MSGCLASS=X,REGION=0M,NOTIFY=&SYSUID
//* TSOBATCH JOB TO RUN REXX IN BATCH *
//*****
-----1-----2-----3-----4-----5-----6-----7-----
//REXXEXEC EXEC PGM=IKJEFT01,PARM='PARMTEST 123456789012345678901234567
//                           8901234567890'
//SYSEXEC   DD DSN=SYS1.LOCAL.EXEC,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD DUMMY
```

© Copyright IBM Corporation 2011

Figure 2-18. PARM continuation and limit

ES074.0

Notes:

The PARM field of the EXEC statement does not allow more than 100 characters, which is an architectural limit. JCL inhibits longer than 100 byte PARMs.

The PARM parameter passes all information between the enclosing parentheses or apostrophes, including commas.

Individual subparameters containing special characters must then be enclosed in apostrophes.

Any continuation character can be put in column 72, and the continuing parms must start in column 16 on the next line.

```
-----1-----2-----3-----4-----5-----6-----7-----
//JCOMP EXEC PGM=BPXBATCH,
// PARM='sh javac -d /u/prichar/prog/java/
//           /u/prichar/prog/java/Fibonacci.java'
//SYSPRINT DD SYSOUT=*
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
```

TIME parameter

- TIME= (*minutes, seconds*)

- On JOB statement
 - Max CPU time for job
- On EXEC statement
 - Max CPU time for step



© Copyright IBM Corporation 2011

Figure 2-19. TIME parameter

ES074.0

Notes:

The TIME parameter can be used to specify the maximum length of processor time that a job or job step is to use the processor.

If coded on the JOB statement, this is the total time for all steps.

If coded on an EXEC statement, it is the maximum time for this step.

Any combination of minutes and seconds can be coded; however, there are three values for the parameter that have a special meaning:

- TIME=NOLIMIT means the job or step is not to be timed.
- TIME=1440 is equivalent to coding TIME=NOLIMIT. Thus, 1440 is the only numerical value of TIME that is not to be interpreted literally.
- TIME=MAXIMUM means the job or step will be allowed to use up to 357,912 minutes. (This is about 8.25 months of processor time. Of course, the job or step will be in the system much longer than this.)

TIME parameter example

```
//JOB JOB TIME=(1,30)
```

Terminate job if total time for all steps exceeds one minute and 30 seconds.

```
//STP1      EXEC PGM=ABLE,TIME=1
```

Terminate this step if time exceeds one minute.

```
//STP2      EXEC PGM=BAKER,TIME=(,20)
```

Terminate this step if time exceeds 20 seconds.

```
//STP3      EXEC PGM=CHARLIE,TIME=NOLIMIT
```

Do not time this step.

© Copyright IBM Corporation 2011

Figure 2-20. TIME parameter example

ES074.0

Notes:

The entire job will be limited to a maximum of one minute 30 seconds. The first step will be limited to a maximum of one minute. The second step will be limited to a maximum of 20 seconds.

Note that the submitter is apparently asking that the last step not be timed; however, coding TIME=NOLIMIT on the last step will *not* accomplish this objective. The TIME=(1,30) on the JOB statement will limit the maximum time for step three to one minute and 30 seconds. What is the minimum time available to step three?

How could the submitter modify the JOB statement time so that step three would not be timed?

If the time consumed by a job (or step) exceeds the value specified for the TIME parameter for that job (or step), the job ABENDs.

EXEC statement examples

```
//STEP1      EXEC PGM=FIRST  
  
//STEPA      EXEC PGM=SECOND, PARM='2003/365', TIME=(,10)  
  
//STEPB      EXEC PGM=THIRD, PARM='12/31/2003'  
  
//STEPD      EXEC PGM=FOURTH, PARM='LIST,MAP,XREF'
```

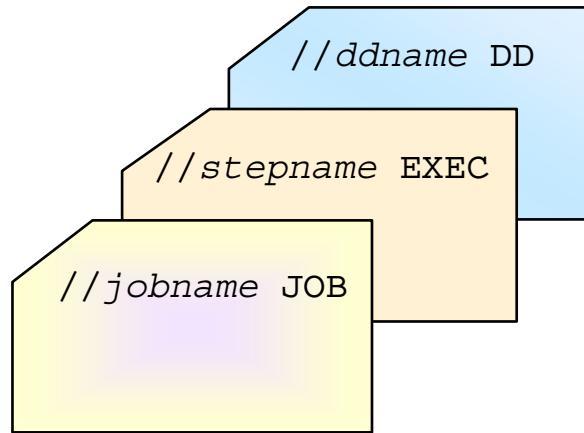
© Copyright IBM Corporation 2011

Figure 2-21. EXEC statement examples

ES074.0

Notes:

The DD statement



© Copyright IBM Corporation 2011

Figure 2-22. The DD statement

ES074.0

Notes:

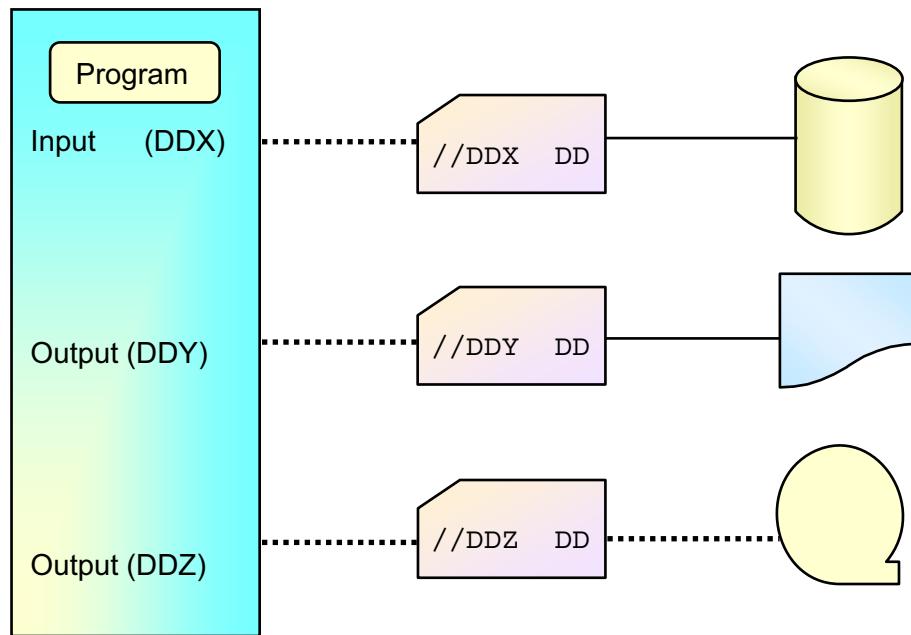
The **DD** statement is used to describe a data set and specify the input and output resources needed for the data set.

A **DD** statement is normally required for each data set that is to be processed in a step.

All **DD** statements for a single step must follow the **EXEC** statement.

DD statements for a single step can usually be in any order.

Why data definition?



© Copyright IBM Corporation 2011

Figure 2-23. Why data definition?

ES074.0

Notes:

The `DD` statement is necessary because the program does not reference a data set directly. The name of a `DD` statement can be coded in the program. When a data set is opened for processing, the name is used to locate the proper `DD` statement.

Most languages allow the programmer complete freedom in the selection of a ddname.

ddnames follow the same rules for all names in JCL:

- One to eight characters in length (alphanumeric or national (#, @, \$)).
- First character must be alphabetic or national.
- Must be followed by at least one blank.

Each `DD` statement should have a unique ddname within a step.

Avoid coding ddnames that begin with `SYS`, `JOB`, or `STEP`.

It is programmer responsibility to document what ddnames can be used with the program, which of them are required and what characteristics are expected.

ddname examples:

COBOL:

```
SELECT IN-FILE ASSIGNING TO DA-3390-S-XYZ -----
```

Assembler:

```
FILE1 DCB DDNAME=XYZ ----->> //XYZ DD ...
```

PL1:

```
DCL XYZ FILE RECORD INPUT -----
```

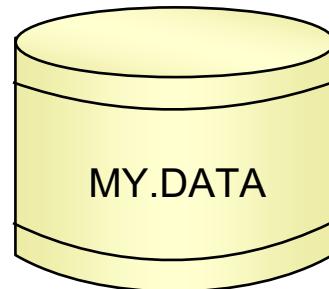
FORTRAN:

```
READ (8,100) A, B ----->> //FT08F001 DD ...
```

Accessing a data set

```
//DD1      DD  DSN=MY.DATA,DISP=OLD
```

References the
data set MY.DATA



© Copyright IBM Corporation 2011

Figure 2-24. Accessing a data set

ES074.0

Notes:

This DD statement would be placed after an EXEC statement that invokes some program. The DD statement would enable the program to access the data set named MY.DATA.

The DD statement has:

- A DDNAME of DD1.
- A data set name of MY.DATA.
- A disposition of OLD. (This just says that the data set already exists.)

Data set concatenation



© Copyright IBM Corporation 2011

Figure 2-25. Data set concatenation

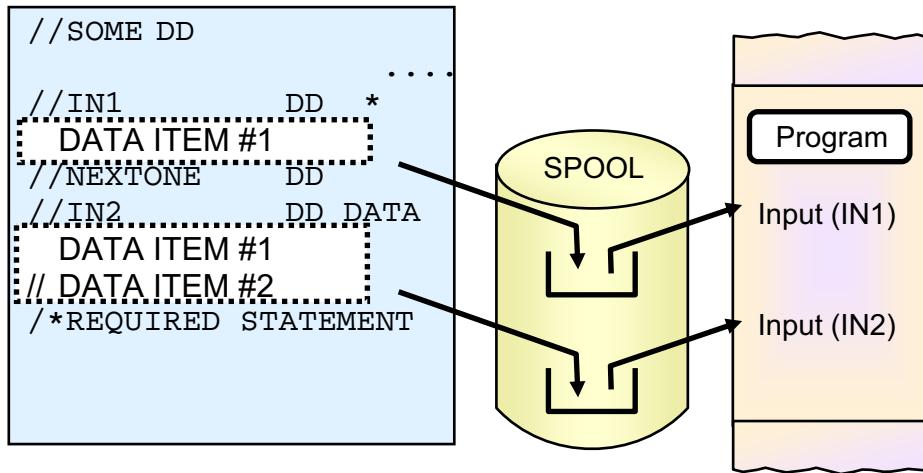
ES074.0

Notes:

An important feature of DD statements is the fact that a single ddname can have multiple DD statements. This is called *concatenation*.

- You can logically connect or *concatenate* sequential or partitioned input data sets (PDSs or PDSEs) in a job or job step.
- The data sets are processed in the order in which they are listed in the concatenation.
- Concatenation does not change the contents of the data sets.
- For sequential data sets:
 - Up to 255 data sets can be concatenated.
 - Data sets can be concatenated in any order of BLKSIZE.
 - DASD and tape data sets can be concatenated.
- For partitioned data sets:
 - A maximum of 123 extents can be concatenated.
 - Data sets can be concatenated in any order of BLKSIZE.

Data in the input stream



© Copyright IBM Corporation 2011

Figure 2-26. Data in the input stream

ES074.0

Notes:

The `DD` statement positional parameter `*` indicates the beginning of an instream data set.

Data records immediately follow the `DD *` and end when one of the following is found:

- `/*` in the input stream
- `//` to indicate another JCL statement
- A two-character delimiter specified on a `DLM` parameter on the `DD` statement
- The input stream runs out of images

The `DD` statement positional parameter `DATA` indicates the beginning of an instream data set that contains statements with `//` in columns 1 and 2.

Data records immediately follow the `DD DATA` and end when one of the following is found:

- `/*` in the input stream
- A two-character delimiter specified on a `DLM` parameter on the `DD` statement
- The input stream runs out of images

Data in the input stream is transferred to Simultaneous Peripheral Operation Online (SPOOL) until it is used by the program.

Instream data set examples:

```
//MYJOB      JOB  
//STEP1      EXEC PGM=IEBGENER  
//SYSPRINT   DD   SYSOUT=A  
//SYSUT1     DD   *
```

```
JOHN DOE    1525 MAIN STREET ANYTOWN STATE 75234 | --This is the  
MARY SMITH  2121 MEADOW LANE OTHERTOWN STATE 75221 | instream data.
```

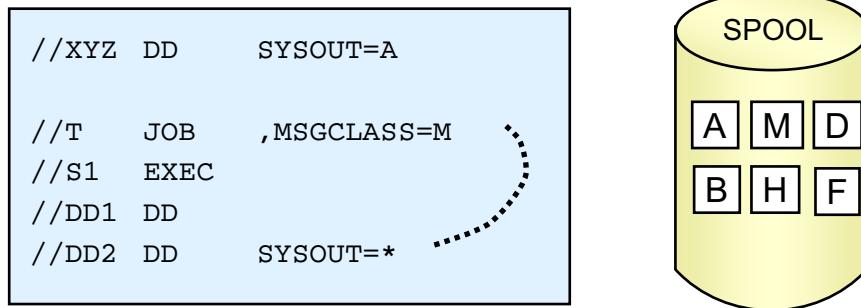
```
//SYSUT2     DD DSN=OUTDATA,DISP=OLD
```

```
//BUILDJOB   JOB  
//STEP1      EXEC PGM=IEBGENER  
//SYSPRINT   DD   SYSOUT=A  
//SYSUT2     DD   DSN=JCL.OUTPUT,DISP=OLD  
//SYSUT1     DD   DATA,DLM=ZZ
```

```
//TEST       JOB   ...          |  
//ST1        EXEC   ...         |  
//INDD       DD    ....        |  
//OUTDD      DD    ....        | -----This is the instream data.  
/*
```

```
ZZ
```

SYSOUT processing



© Copyright IBM Corporation 2011

Figure 2-27. SYSOUT processing

ES074.0

Notes:

The `SYSOUT` parameter requests that JES write data to the SPOOL. The data is then normally printed or viewed through a product such as System Display and Search Facility (SDSF).

The format is `SYSOUT=class`.

- Class values can be one character, A to Z, 0 to 9, or *.
- It is a common practice to designate class A and B as follows:
 - A System printer (local) with standard forms.
 - B System punch (local).

`SYSOUT=*` uses the same class as `MSGCLASS`. If the class used by `MSGCLASS` is also a held class and if all `SYSOUT` parameters use '*' for the class, the entire job output is held.

SYSOUT queuing example:

		A	C	SYSOUT QUEUES NETWORK
//XYZ JOB ...				
//S1 EXEC PGM=ONE		XYZ	XYZ	XYZ
//DD1 DD SYSOUT=A,COPIES=2		STD	STD	TSO99
//DD2 DD SYSOUT=C				
//DD3 DD SYSOUT=(A,,FM29)				
//S2 EXEC PGM=TWO		XYZ	XYZ	
//DD4 DD SYSOUT=(C,,FM29)		FM29	FM29	
//DD5 DD SYSOUT=A,DEST=TSO99				

COMMENT statement

- If //* is coded in positions 1, 2, and 3 of a JCL statement, the entire statement is considered to be a comment.

```

Submitted JCL
//*****  

//** THE FOLLOWING IS AN EXAMPLE OF A BAD CONTINUATION ***}  

//*****  

//BADJOB JOB (378,'6/10),DICK,MSGLEVEL=(1,1)  

//          CLASS=A,NOTIFY=TSOMJ02  

//STEPX      EXEC .....

Listing
//*****  

//** THE FOLLOWING IS AN EXAMPLE OF A BAD CONTINUATION **}  

//*****  

//BADJOB JOB (378,'6/10),DICK,MSGLEVEL=(1,1)  

//          CLASS=A,NOTIFY=TSOMJ02  

//STEPX      EXEC .....

```

The diagram shows two boxes labeled "Submitted JCL" and "Listing". Inside each box is a block of JCL code. Above the code in each box is a brace that spans the entire length of the code, indicating that the code in both boxes is identical. Arrows point from the labels "Submitted JCL" and "Listing" to their respective boxes.

© Copyright IBM Corporation 2011

Figure 2-28. COMMENT statement

ES074.0

Notes:

Columns 1, 2, and 3 of the *comment statement* are coded //*. Code comments in columns 4 through 80. Note that the COMMENT statement does not contain the four fields present on most JCL statements.

The system signals how it processed your job statements using a three-character code in positions 1, 2, and 3 of a line in the JCL listing. These codes are:

- /* Any JES2 statement
- /** Any JES3 statement
- /** Any comment statement
- // Submitted JCL statement

Checkpoint (1 of 3)

1. Which of the following are valid JOB statement keyword parameters?
 - a. REGION
 - b. CLASS
 - c. NOTIFY
 - d. COMMAND
2. Which keyword defines the job log output class?
3. True or False: Statements are coded from column 1 to column 72.
4. True or False: A continuation of a statement must start after column 16.

© Copyright IBM Corporation 2011

Figure 2-29. Checkpoint (1 of 3)

ES074.0

Notes:

Checkpoint (2 of 3)

5. True or False: STEP names must be unique in a JOB.
6. True or False: Key words can be in both uppercase and lowercase.
7. True or False: REGION=0M means you can allocate as much as you want above 2 GB.
8. True or False: MEMLIMIT=0 means you cannot allocate storage above 2 GB.

© Copyright IBM Corporation 2011

Figure 2-30. Checkpoint (2 of 3)

ES074.0

Notes:

Checkpoint (3 of 3)

9. True or False: TIME=1440 means the job cannot execute longer than 24 hours (24x60).
10. What is the DD statement to indicate data in the input stream?
11. Which keyword tells the system to automatically insert your user ID here so that the information will be sent to you?
 - a. NOTIFY=SY&UID
 - b. NOTIFY=&SYSID
 - c. NOTIFY=&SYSUID
12. Job name length can be:
 - a. One to seven characters
 - b. One to eight characters
 - c. One to 10 characters

© Copyright IBM Corporation 2011

Figure 2-31. Checkpoint (3 of 3)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Describe the JCL statement format
- Code JOB statements and JOB statement parameters
- Code EXEC statements to invoke a program and pass PARM parameters to the program
- Introduce the DD statement and a few of its parameters
- Discuss the COMMENT statement

© Copyright IBM Corporation 2011

Figure 2-32. Unit summary

ES074.0

Notes:

Unit 3. DD parameters: A second look

What this unit is about

The `DD` statement is used to describe a data set and to specify the input and output resources needed for the data set. The `DD` statement consists of positional or keyword parameters or both. All keyword parameters are optional; this unit looks at the keywords that are most often used.

Special `DD` statements are also introduced.

What you should be able to do

After completing this unit, you should be able to:

- Create a new data set
- Reference an existing uncataloged data set
- Reference an existing cataloged data set
- Code and discuss the `DISP` parameter
- Use special `DD` statements
- Detect and correct JCL syntax and usage errors

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597	<i>z/OS MVS JCL Reference</i>
SA22-7598	<i>z/OS MVS JCL User's Guide</i>

Unit objectives

After completing this unit, you should be able to:

- Create a new data set
- Reference an existing uncataloged data set
- Reference an existing cataloged data set
- Code and discuss the `DISP` parameter
- Use special `DD` statements
- Detect and correct JCL syntax and usage errors

© Copyright IBM Corporation 2011

Figure 3-1. Unit objectives

ES074.0

Notes:

DD statement syntax

//ddname DD *positional, keywords*

```
//XYZ DD DSN=___,DISP=__,
//      UNIT=___,SPACE=___,VOL=__
```



© Copyright IBM Corporation 2011

Figure 3-2. DD statement syntax

ES074.0

Notes:

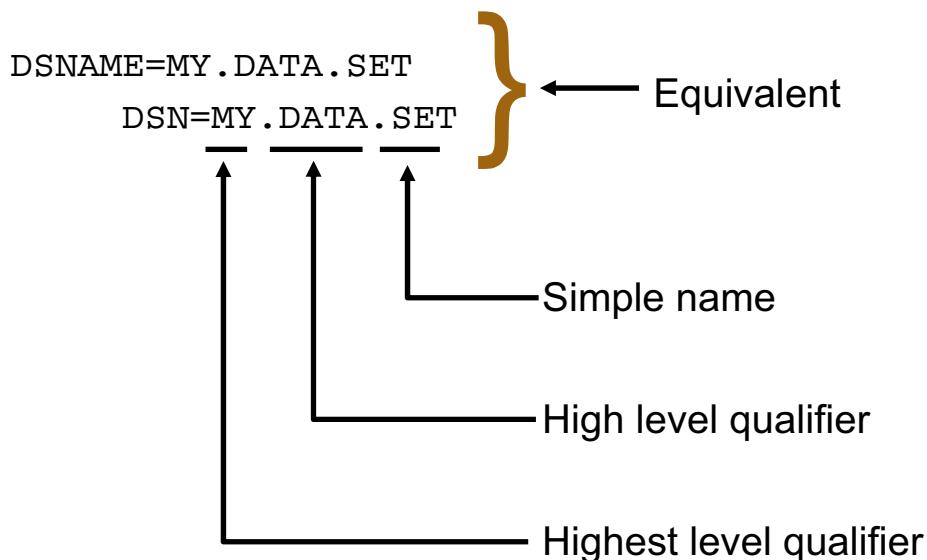
Either positional or keyword parameters can be coded on a `DD` statement, but keyword parameters are probably the most common.

Over 50 keyword parameters are available, but fortunately you rarely have to code more than half a dozen on any one `DD` statement. You typically select and code the parameters which are appropriate for the situation at hand.

Five of the most commonly coded keyword parameters are shown above. Some parameters are rather general in nature. For example, it is appropriate to code `DISP` on almost any `DD` statement. On the other hand some parameters are specialized in nature. For example, `SPACE` is used only in dealing with DASD data sets.

However, this class will talk about a number of other `DD` statement parameters. Refer to *z/OS MVS JCL Reference* for the complete list of parameters.

Permanent data set naming



The DSN can be a maximum of 44 characters.

© Copyright IBM Corporation 2011

Figure 3-3. Permanent data set naming

ES074.0

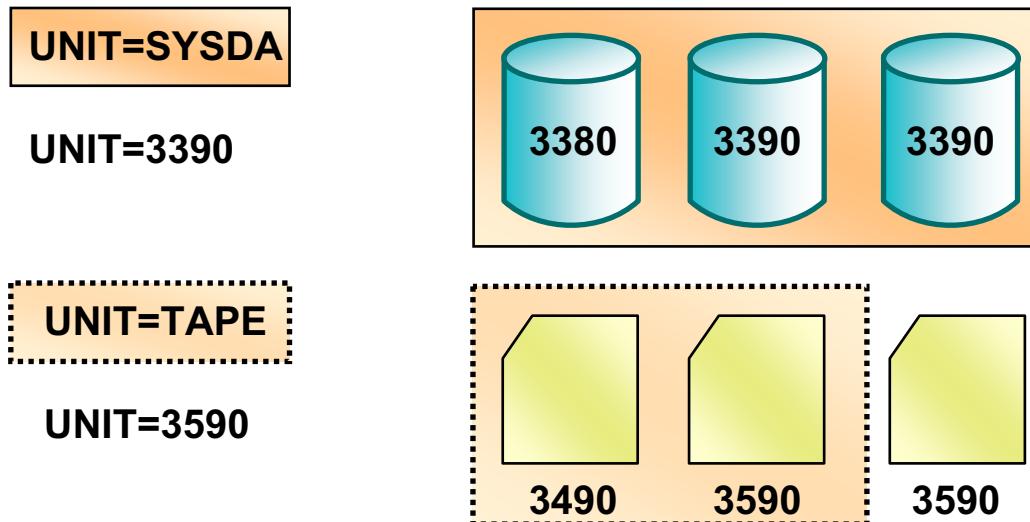
Notes:

DSN= value is the name of the data set; this can include creation of temporary data sets or a reference back to the data set name.

DSNAME= or **DSN=** parameter value follows these conventions:

- Simple names are one to eight alphanumeric or national (#, @, \$) characters
- First character must be alphabetic or national
- Join simple names together with periods to form qualified names
- Qualified names can be a maximum of 44 characters

Unit selection



© Copyright IBM Corporation 2011

Figure 3-4. Unit selection

ES074.0

Notes:

UNIT= value indicates system disk, tape, special device type, or esoteric (local name).

This parameter identifies the device or type of device on which the data set will be allocated. Use the UNIT parameter to ask the system to place the data set on:

- A specific device
- A certain type or group of devices
- The same device as another data set

You use this parameter to specify the number of devices to be used. If the data set exists, you only need to specify the device type if the data is not cataloged. Most installations now administer disk storage with the Data Facility Storage Management Subsystem/Multiple Virtual Storage (DFSMS/MVS). With SMS, you do not need to use the UNIT parameter to specify a device for SMS-controlled data sets.

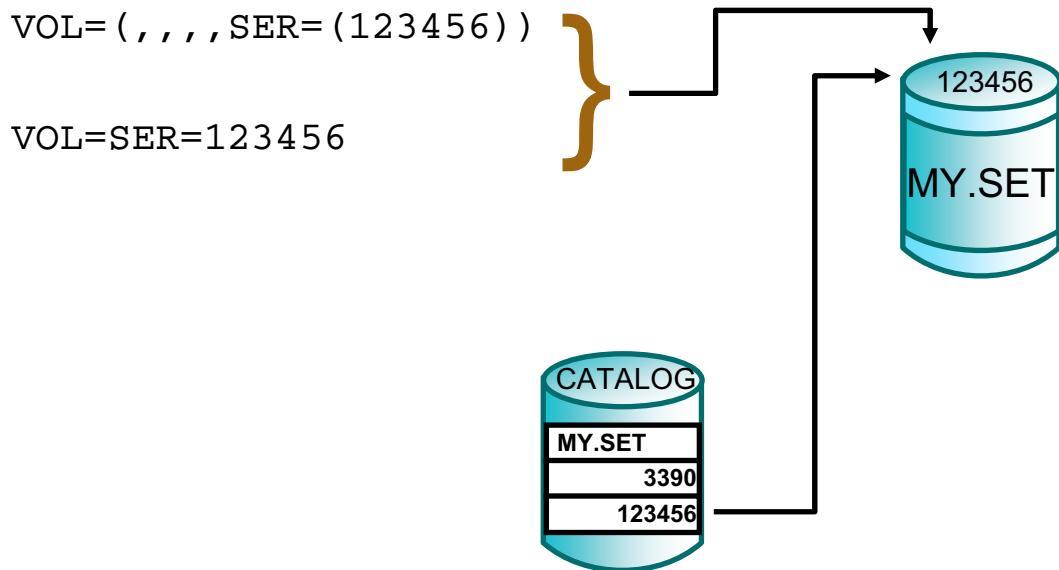
Some common examples are:

UNIT=SYSDA	Allocates the data set on a direct access storage device (DASD)
------------	---

UNIT=3390	Allocates the data set on a 3390 type disk
UNIT=SYSALLDA	Allocates the data set on a direct access storage device (DASD)
UNIT=TAPE	Allocates the file on a TAPE device

- Three types of UNIT requests can be made. Here are three examples:
 - Demand requests: UNIT=/4FC0 where 4FC0 is a device number
 - Generic requests: UNIT=3390 where 3390 is an eligible device
 - Esoteric requests: UNIT=SYSDA where SYSDA is associated with DASD
- An esoteric name is an installation-specified name which can be assigned to multiple device types. The esoteric names SYSDA and SYSSQ are often used in z/OS installations. SYSDA usually includes all DASD units and SYSSQ usually includes all tape and DASD units.
- Esoteric requests are generally preferred over generic requests, and generic requests are generally preferred over demand requests.
- The UNIT parameter can also request a specific number of devices to assign and can request that the operator defer mounting of the volume until the data set is opened.

Volume specification



© Copyright IBM Corporation 2011

Figure 3-5. Volume specification

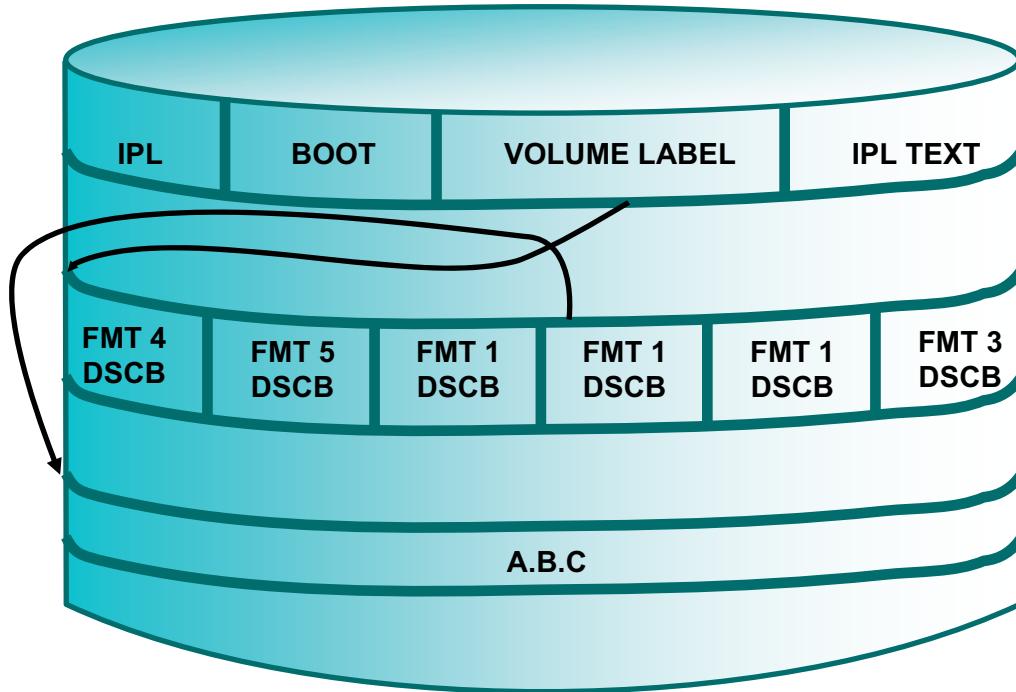
ES074.0

Notes:

VOL=SER= value specifies volume name, disk name, or tape name. Use VOLUME parameter to identify the volume or volumes on which an existing data set resides or a new data set will reside.

- VOL and VOLUME can be used interchangeably.
The full syntax for coding the VOL parameter is as follows.
`VOL= ([PRIVATE] , [RETAIN] , Seq#, Count , SER=serial_number)`
PRIVATE: requests a private volume. Code VOL=SER to request the volume.
RETAIN: applies only to tape. Do not demount tape at close of data set.
Seq#: specifies the volume on which processing is to begin.
Count: specifies the maximum number of volumes for an output data set.
serial_number: allows SER to be coded. See the example above.
- Request multiple volumes by using the format `VOL=SER= (____,____,____,...)` where serial numbers are supplied in place of the blanks.
- `VOL=REF=____` can be coded for non-SMS managed data sets. It should not be used for SMS-managed data sets. If `VOL=REF=X.Y.Z` is coded, then X.Y.Z must be a cataloged data set or passed from some preceding step in the job.

DASD volume table of contents



© Copyright IBM Corporation 2011

Figure 3-6. DASD volume table of contents

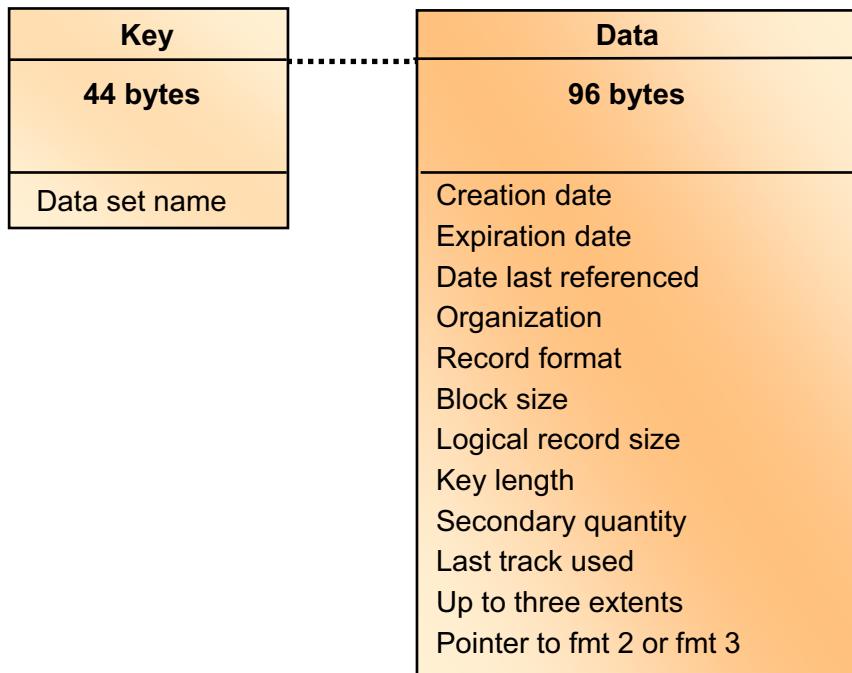
ES074.0

Notes:

z/OS uses a catalog and a volume table of contents (VTOC) on each DASD to manage the storage and placement of data sets:

- All DASD volumes have a standard volume label that contains the volume serial number and pointer to the VTOC.
- The VTOC is a physical sequential file with different record formats.
- Data set control blocks (DSCBs) are the records that make up a VTOC.
- There are seven DSCB formats:
 - Format 0: This represents an available DSCB in the VTOC.
 - Format 1: This describes an existing DASD data set.
 - Format 2: Indexed sequential access method (ISAM) uses this to describe ISAM index areas and options.
 - Format 3: This describes additional (up to 13) data set extents.
 - Format 4: This describes the VTOC. It appears first in the VTOC.
 - Format 5: This describes available space on the volume.
 - Format 6: This describes split cylinder allocations. (This is not used by z/OS.)

DSCB content (format 1)



© Copyright IBM Corporation 2011

Figure 3-7. DSCB content (format 1)

ES074.0

Notes:

The VTOC is composed of 140-byte data set control blocks (DSCBs) that correspond either to a data set or Virtual Storage Access Method (VSAM) data space currently residing on the volume or to contiguous, unassigned tracks on the volume.

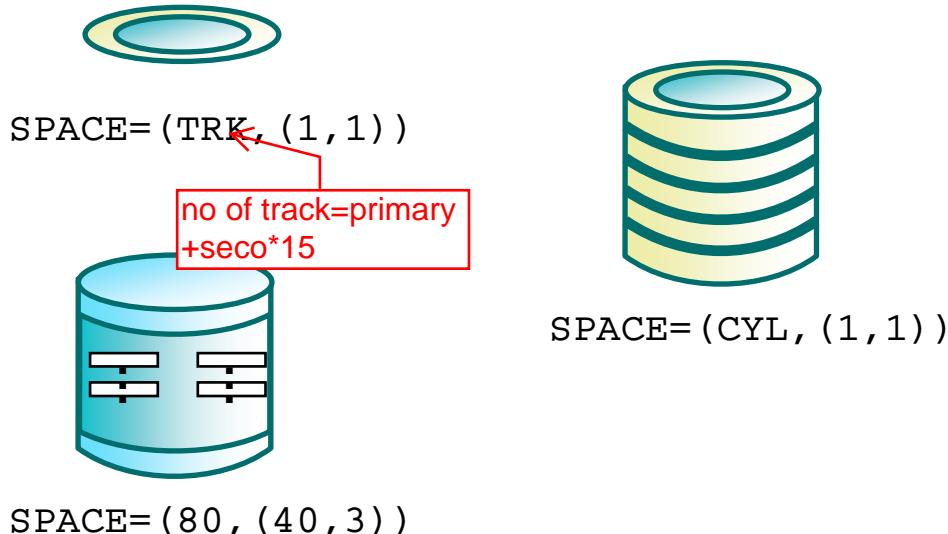
DSCB is the name of the logical record within the VTOC. DSCBs describe data sets allocated on that volume and also describe the VTOC itself. The system automatically constructs a DSCB when space is requested for a data set on a direct access volume. Each data set on a DASD volume has one or more DSCBs to describe its characteristics.

- **Key:** A 44-byte key containing the data set name.
- **Data:** A 96-byte data area follows the key on the track.

Space specification

Sequential data set

`SPACE= (Unit, (Primary, Secondary) , __)`



© Copyright IBM Corporation 2011

Figure 3-8. Space specification

ES074.0

Notes:

The `SPACE` DD parameter is required for allocating data sets on DASD. It identifies the space allocation required for your data set. Before a data set can be created on disk, the system must know how much space the data set will require and how the space is to be measured.

- Code the `SPACE` parameter to request space for a new data set on a direct access volume.
- The format for the space parameter is as follows:

`SPACE= (UNIT OF ALLOCATION, (PRIMARY, SECONDARY))`

Unit of allocation:

- **TRK**: Requests that space be allocated in tracks.
- **CYL**: Requests that space be allocated in cylinders.
- **blklgth (*only if AVGREC is not coded*)**: Specifies the average block length, in bytes, of the data. The system calculates the number of tracks to allocate.
- **reclgth (*only if AVGREC is coded*)**: With SMS, specifies the average record length, in bytes, of the data. The system calculates the number of tracks to allocate.

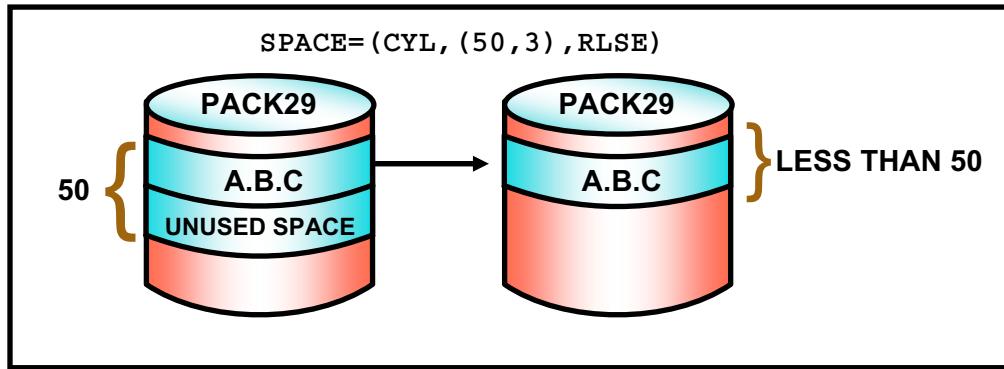
How many:

- **Primary**: Primary amount of space for allocation. Allocated one time only. The primary quantity must be available on the volume or the allocation fails.
- **Secondary**: Specifies the number of additional tracks, cylinders, blocks, or records to be allocated if more space is needed. Allocated up to 15 times.

Some examples of how you can code:

- **SPACE= (TRK, 4)** : To allocate space to a sequential data set, requesting four tracks, only primary space. After 4 tracks are used, the data set cannot grow anymore.
- **SPACE= (CYL, (5, 2))** : To allocate space to a sequential data set, five cylinders as primary allocation space and two cylinders as secondary.
- **SPACE= (CYL, (10, ,100))** : To allocate space to a partitioned data set, only primary allocation of 100 cylinders and 100 directory blocks.
- **SPACE= (TRK, 10)** : Ten tracks, no secondary extents.
- **SPACE= (TRK, (10, 5))** : Ten tracks primary, five tracks for each secondary extent
- **SPACE= (CYL, 5)** : Can use CYL (cylinders) instead of TRK
- **SPACE= (TRK, (10, 5, 8))** : PDS with eight directory blocks
- **SPACE= (1000, (50,000,10000))** : Primary 50000 records @ 1000 bytes each

Release unused DASD space



© Copyright IBM Corporation 2011

Figure 3-9. Release unused DASD space

ES074.0

Notes:

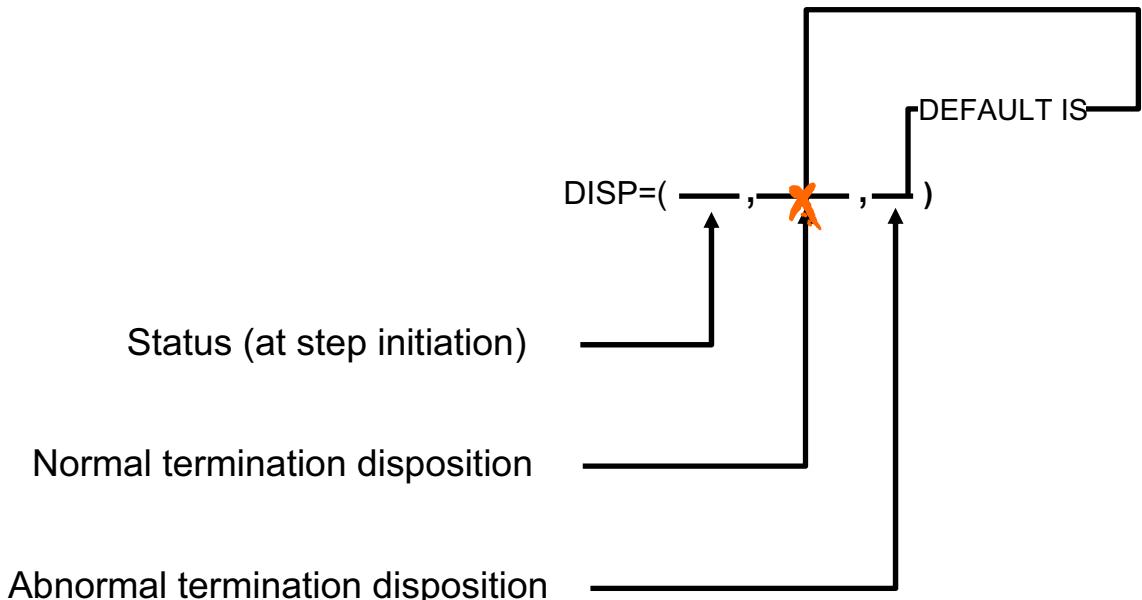
Code the `RLSE` parameter to release any allocated, unused space when the data set is closed.

If you specify `RLSE` and an abnormal termination occurs, the system does not release unused space.

RLSE example:

```
//MYDD      DD    DSN=A.B.C,DISP=(,CATLG),
//          SPACE=(CYL,(50,3),RLSE)
```

DISP: Syntax and defaults



© Copyright IBM Corporation 2011

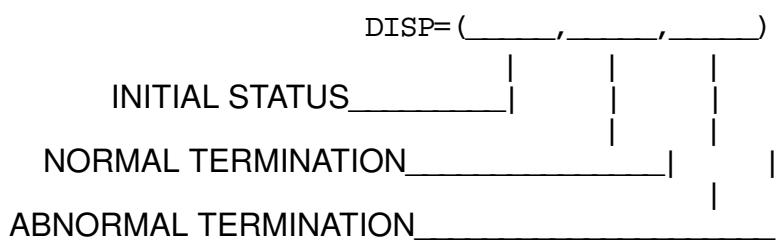
Figure 3-10. DISP: Syntax and defaults

ES074.0

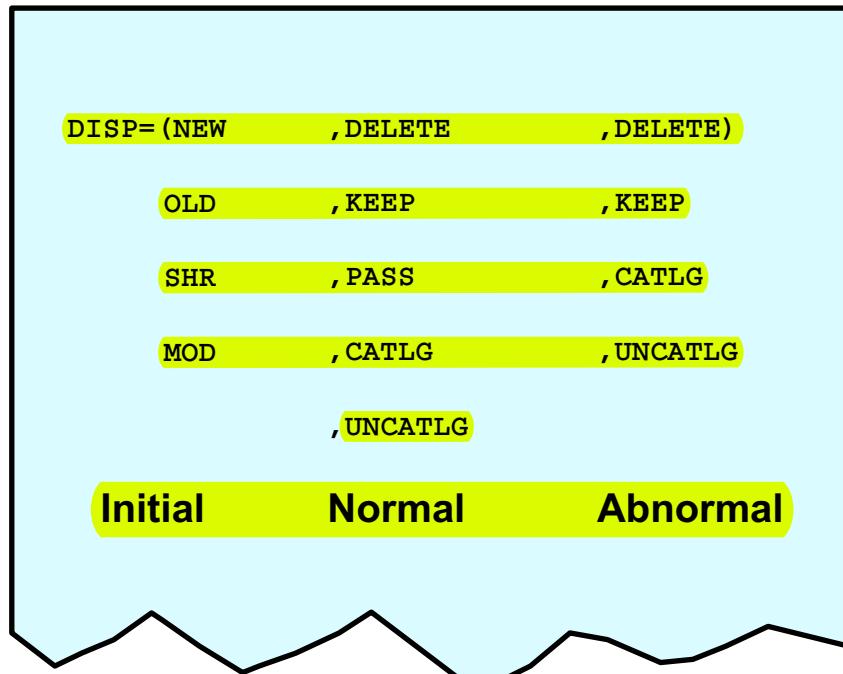
Notes:

The disposition (DISP) parameter on the DD statement describes the status of the data set at step initiation, normal step end, and abnormal step end. All subparameters within the DISP parameter are positional.

- *Initial* is the status of the data set at the beginning of the step.
 - *Normal* termination is the disposition of the data set when the job step terminates normally.
 - *Abnormal* termination is the disposition of the data set when the job step terminates abnormally (that is, ABEND).



DISP parameter



© Copyright IBM Corporation 2011

Figure 3-11. DISP parameter

ES074.0

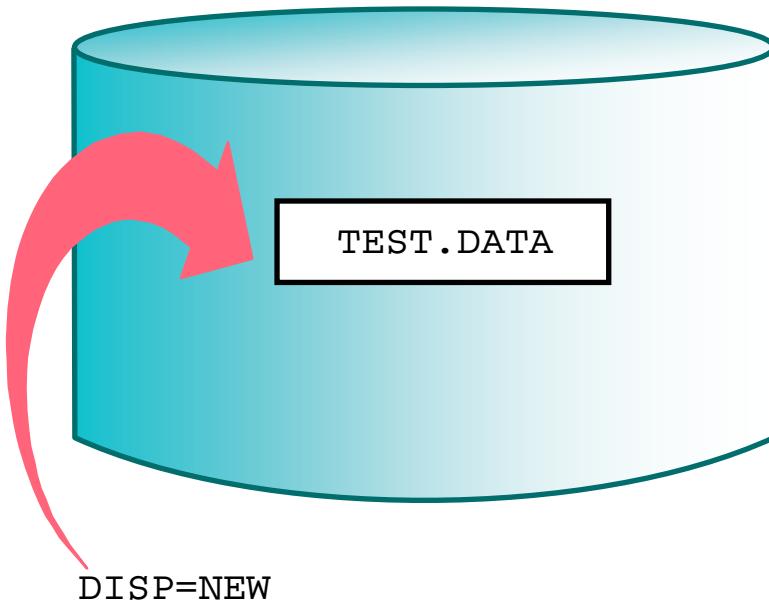
Notes:

Displayed above is the disposition parameter with the subparameter values that can be coded for each status.

DISP parameter defaults:

NO DISP	DISP= (NEW, DELETE, DELETE)
DISP=OLD	DISP= (OLD, KEEP, KEEP)
DISP=(, CATLG)	DISP= (NEW, CATLG, CATLG)
DISP=NEW	DISP= (NEW, DELETE, DELETE)
DISP=SHR	DISP= (SHR, KEEP, KEEP)

DISP processing: NEW



© Copyright IBM Corporation 2011

Figure 3-12. DISP processing: NEW

ES074.0

Notes:

The first field identifies the status of the data set and how to control access to it.

- When the first field is NEW, it indicates that the data set will be created and the job will have exclusive control of the data set.
- This means that no other job can access this data set until the last step in this job that refers to this data set ends.
- If the data set already exists, the job will be terminated.
- NEW is the default.

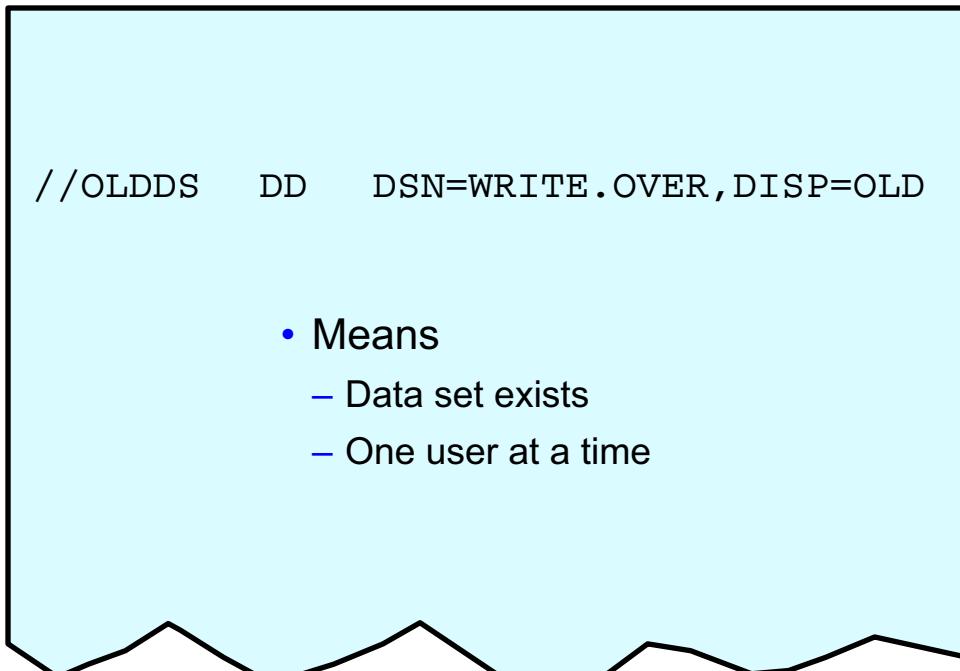
Code DISP=NEW when creating a new data set.

DISP=NEW allows only one user to access the data set at a time.

DISP processing: OLD

```
//OLDDDS    DD    DSN=WRITE.OVER,DISP=OLD
```

- Means
 - Data set exists
 - One user at a time



© Copyright IBM Corporation 2011

Figure 3-13. DISP processing: OLD

ES074.0

Notes:

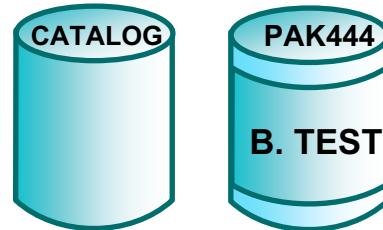
DISP=OLD indicates the data set exists and the job requires exclusive access to it.

- Code DISP=OLD to reference an existing data set.
- DISP=OLD allows only one user to access the data set at a time. The system places other users in a wait.
- DISP=OLD requires the data set to exist before the step executes. Otherwise the job is terminated.
- If DISP=OLD is coded on output, the system starts at the beginning of the data set and writes over existing data. The system writes an end-of-file (EOF) after writing the last record.

DISP processing: KEEP

Creation

```
//DCZ DD      DSN=B.TEST,UNIT=SYSDA,
//      SPACE=(CYL,(10,5)),
//      DISP=(NEW,KEEP)
```



Access

```
//DCZ DD      DSN=B.TEST,UNIT=SYSDA,VOL=SER=PAK444,
//      DISP=OLD
```

© Copyright IBM Corporation 2011

Figure 3-14. DISP processing: KEEP

ES074.0

Notes:

The `DISP` parameter value `NEW` indicates the data set does not exist and is to be created.

The `DISP` parameter value `KEEP` indicates the data set will be kept at the end of step.

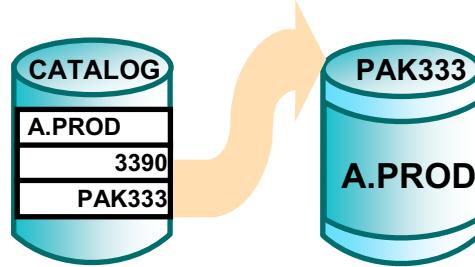
Note that `VOL=SER` was omitted from the creating `DD` statement.

- Since it was omitted, the system will pick the volume. Thus, if there is insufficient space on the first volume selected, the system continues to hunt until it finds a volume with enough space (if there is an eligible volume with enough space).
- This is an example of a nonspecific volume request. A nonspecific volume request is one in which the `VOL=SER` has been omitted (and not implied) on the `DD` statement that requests the creation of the data set.
- However, you will have to wait until the job completes to find out what volume the system picked. After you find the volume, then you can code the accessing `DD` statement.
- Both the `UNIT` and `VOL=SER` must be coded when you submit another job to access the data set. Why?

DISP processing: CATLG

Creation

```
//DC1 DD      DSN=A.PROD,UNIT=SYSDA,SPACE=(80,(300)),
//          VOL=SER=PAK333,
//          DISP=(NEW,CATLG)
```



Access

```
//DA1 DD      DSN=A.PROD,DISP=OLD
```

© Copyright IBM Corporation 2011

Figure 3-15. DISP processing: CATLG

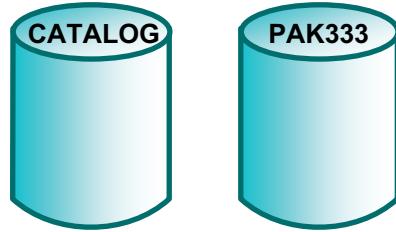
ES074.0

Notes:

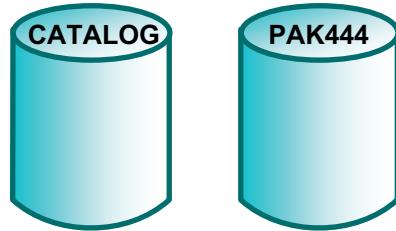
- The `DISP` parameter value `NEW` indicates that the data set does not exist and is to be created.
- The `DISP` parameter value `CATLG` indicates:
 - The data set will be kept at the end of the step.
 - The `UNIT` and `VOL=SER` information will be recorded in the catalog. That is, the system will remember the data set location for you.
- Note that `VOL=SER` was coded on the creating `DD` statement.
 - Since it was coded, you will know ahead of time where the data set will be placed.
 - This is an example of a *specific* volume request. A specific volume request is one in which the `VOL=SER` has been coded (or implied) on the `DD` statement that requests the creation of the data set.
 - Both the `UNIT` and `VOL=SER` can be omitted when you submit another job to access the data set. Why?
 - You could code both the `UNIT` and `VOL=SER` when you submit another job to access the data set, but you should avoid the temptation. Why?

DISP processing: DELETE

```
//DC1 DD DSN=A.PROD,
//      DISP=(OLD,DELETE)
```



```
//DL2 DD DSN=B.TEST,
//      UNIT=SYSDA, VOL=SER=PAK444,
//      DIS=(OLD,DELETE)
```



© Copyright IBM Corporation 2011

Figure 3-16. DISP processing: DELETE

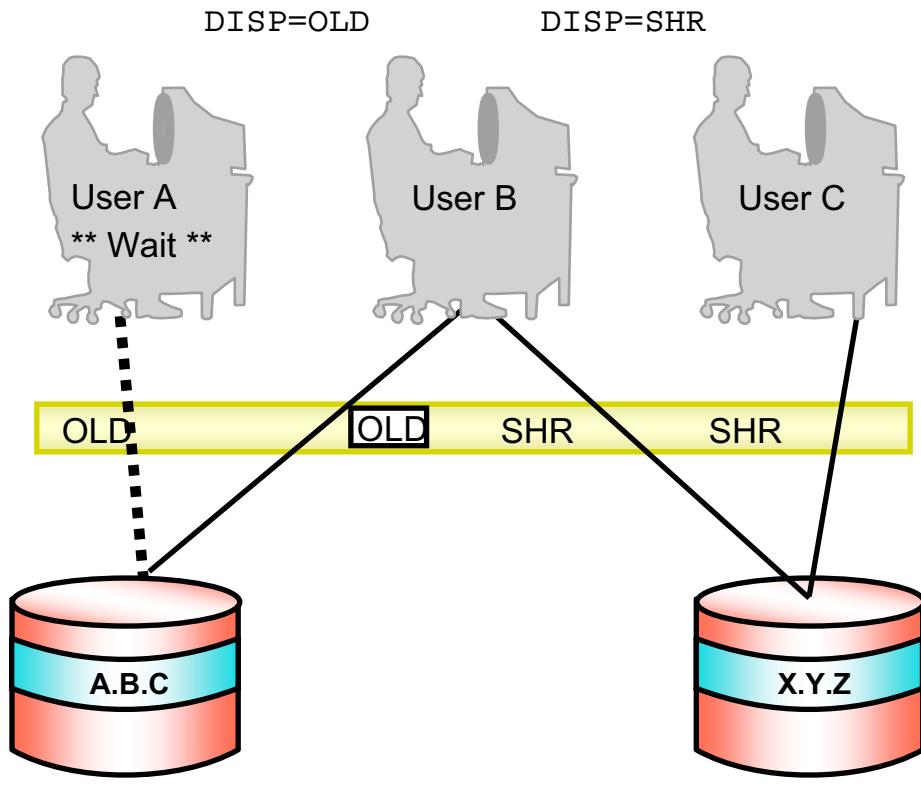
ES074.0

Notes:

Since the data set B.TEST was never cataloged, you must code UNIT and VOL=SER to delete it. Otherwise, the system will not know where it is.

Since the data set A.PROD was cataloged, you can omit UNIT and VOL=SER when you delete it. (The system knows where it is.) In fact you should omit UNIT and VOL=SER in this case. If you omit the parameters, the system will delete the data set and delete the entry from the catalog. On the other hand, if you code both parameters, the system will delete the data set but not delete the catalog entry. Thus someone will have to clean up the catalog at a later date.

DISP: Data set sharing



© Copyright IBM Corporation 2011

Figure 3-17. DISP: Data set sharing

ES074.0

Notes:

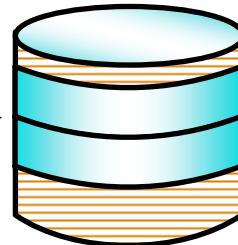
`SHR` indicates that the data set can be shared by other users.

Code `DISP=SHR` to reference an existing data set and allow other users to access the data set at the same time.

DISP processing: MOD

```
//MOD DD DSN=ADD.TO.END,DISP=(MOD,PASS),UNIT=SYSDA,SPACE=(CYL,(3,1))
```

- If data set exists:
 - Add records to logical end of data



© Copyright IBM Corporation 2011

Figure 3-18. DISP processing: MOD

ES074.0

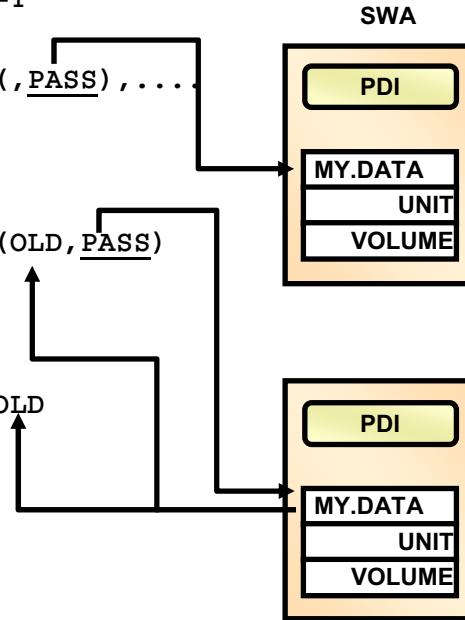
Notes:

MOD indicates that if the data set exists, data will be appended to the end of the data set; otherwise, a new data set will be created. The job requires exclusive access to the data set.

- Code `DISP=MOD` to indicate records are to be appended to the logical end of an existing data set.
- If `DISP=MOD` is coded and the data set does not exist, the disposition is treated as `DISP=NEW`.

DISP processing: PASS

```
//SAMPLE      JOB 99,IBMUSER,CLASS=T
//ONE        EXEC PGM=FIRST
//DXX        DD DSN=MY.DATA,DISP=(,PASS),...
...
...
//TWO        EXEC PGM=SECOND
//DDY        DD DSN=MY.DATA,DISP=(OLD,PASS)
...
...
//THREE      EXEC PGM=FOUR
//DDZ        DD DSN=MY.DATA,DISP=OLD
```



© Copyright IBM Corporation 2011

Figure 3-19. DISP processing: PASS

ES074.0

Notes:

DISP=PASS

- Code PASS to indicate the data set is to be passed for use by a subsequent step in the same job.
- PASS is effective only within a job.
- Code DISP=(____, PASS) on each reference.
- The passed data set information (PDI) entry is created at step termination in response to the PASS disposition.
- The PDI is a queue of control blocks kept in the scheduler work area (SWA). The PDI closely resembles a catalog entry.
- If a final normal termination DISP is not coded, the defaults are:
 - Data set existed before the job: Data set is retained
 - Data set created in the job: Data set is deleted

In the following example, the catalog is used to locate the data set in STEPX.
Subsequently, the STEPY reference will use the PDI instead of saving a catalog search.

PASS example:

```
//STEPX EXEC PGM=ANY1
//OLDDD DD DSN=ABC,DISP=(OLD,PASS) System uses the catalog.
//STEPY EXEC PGM=ANY2
//OLDDD DD DSN=ABC,DISP=(OLD,PASS) System uses PDI.
```

Backward reference: Example

DSN= { * .ddname
 * .stepname .ddname

```
//BREF      JOB    789 ,MARY,CLASS=A
//ONE       EXEC    PGM=ABLE
//MAST      DD      DSN=A.B.C,DISP=OLD
//OPUT      DD      DSN=X.Y.Z,DISP=(,KEEP),
//           UNIT=SYSDA,SPACE=(CYL,2)
//M2        DD      DSN=*.MAST,DISP=OLD
                           ...
//TWO        EXEC    PGM=BAKER
//MAST      DD      DSN=*.ONE.MAST,DISP=OLD
//MOD       DD      DSN=*.ONE.OPUT,DISP=(MOD,CATLG),
//           VOL=REF=*.ONE.OPUT
```

© Copyright IBM Corporation 2011

Figure 3-20. Backward reference: Example

ES074.0

Notes:

The presence of the '*' in the visual indicates that a backward reference is being used.

- Many parameters in JCL statements can use a backward reference to fill in information. A backward reference (or *referback*) is a reference to an earlier statement in the job or in a cataloged or instream procedure called by a job step. A backward reference has the form:

*.NAME or *.DDNAME

*.STEPNAME.NAME or *.STEPNAME.DDNAME

*.STEPNAME.PROCSTEPNAME.NAME or *.STEPNAME.PROCSTEPNAME.DDNAME

- Step names *must* be unique to use backward references.

Temporary (work) data sets

OMIT DSN

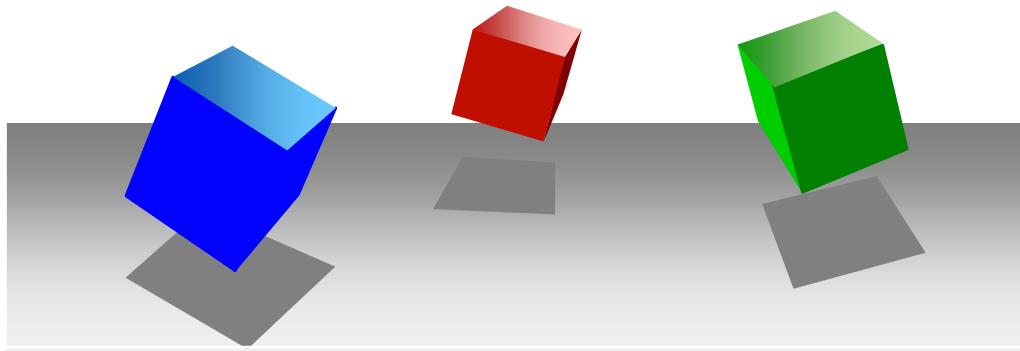
```
//A DD UNIT=SYSDA, SPACE=(CYL, 1)
```

DSN=&&WORK

```
//B DD DSN=&&WORK, UNIT=SYSDA, SPACE=(CYL, 1)
```

DSN=&TEMP

```
//C DD DSN=&TEMP, UNIT=SYSDA, SPACE=(CYL, 1)
```



© Copyright IBM Corporation 2011

Figure 3-21. Temporary (work) data sets

ES074.0

Notes:

A temporary data set is one that is created and deleted in the same job.

There are three ways to request the creation of a temporary data set:

- Omit the DSN. In this case, the system will create a temporary data set having a 44-character name. The system builds date, time, and other information into this name to ensure the name is unique.
- Code a name beginning with &&, such as DSN=&&WORK. The system will create a temporary data set having a 44-character name. The system builds date, time, and other information into the name to ensure its uniqueness.
- Code a name beginning with &, such as DSN=&TEMP. The system will create a temporary data set with a 44-character name. The system creates a unique name containing date, time and other information. Names prefixed by && are preferable to those prefixed by &. In a name of the form, DSN=&TEMP, the &TEMP can mean a temporary data set or a symbolic parameter (to be covered later). To avoid ambiguity use the form DSN=&&TEMP.

If you use two temporary data sets of the forms DSN=&&WORK and DSN=&TEMP in the same job stream, the system will interpret these to be the same data set.

Temporary data sets: Example

```

//TJOB    JOB          3344,WDRK,CLASS=T
//STEP1   EXEC      PGM=ABC
//WK      DD          UNIT=SYSSQ,SPACE=(4096,(300,100))
//WK1     DD          UNIT=3390,SPACE=(CYL,(1,1)),DISP=(,PASS)
//WK2     DD          DSN=&&WK2,UNIT=SYSDA,SPACE=(4096,(100,20)),
//                  DISP=(,PASS)
      ....
//STEP2   EXEC      PGM=DEF
//WORKIN  DD          DSN=&&WK2,DISP=(OLD,DELETE)
//WORKIN2 DD          DSN=*.STEP1.WK1,DISP=(OLD,PASS)

```

The diagram illustrates the creation of temporary data sets. It shows a horizontal track with three colored rectangular blocks: yellow, blue, and red. Below the track, there is JCL code. The first two DD statements in STEP1 create temporary data sets (yellow and blue), which are then referenced by the first DD statement in STEP2. The third DD statement in STEP1 creates another temporary data set (red), which is referenced by the second DD statement in STEP2.

© Copyright IBM Corporation 2011

Figure 3-22. Temporary data sets: Example

ES074.0

Notes:

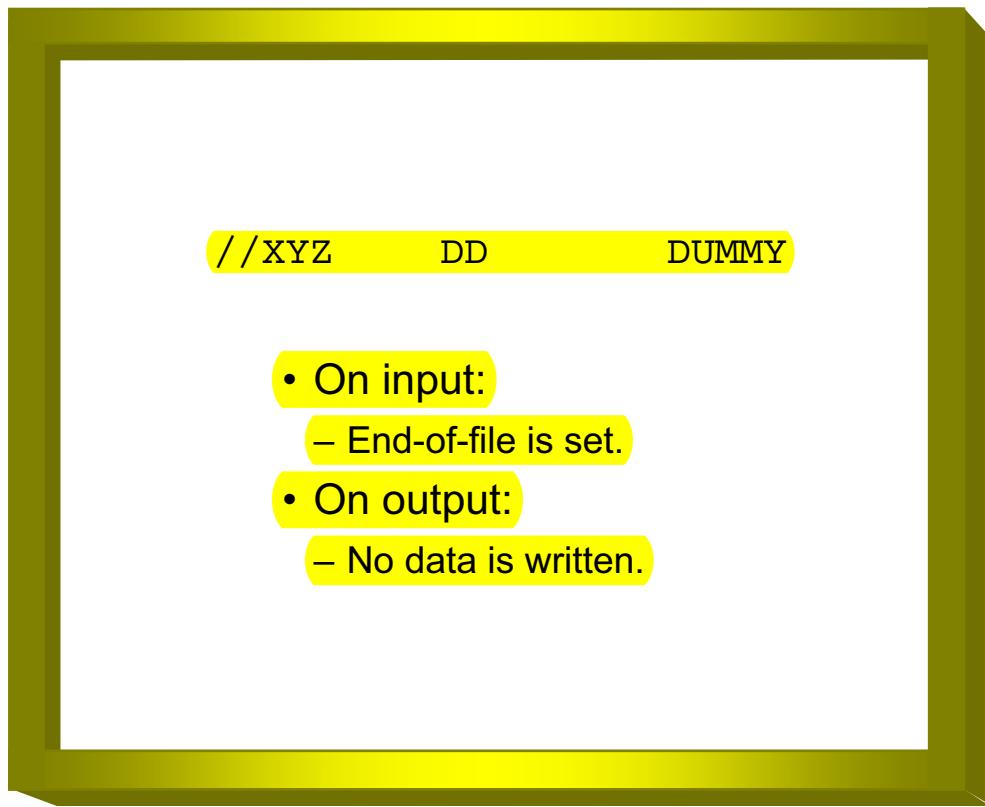
Note that three different temporary data sets are created for the three `DD` statements in `STEP1`.

- The `DSN` parameter has been omitted from the first two `DD` statements, so z/OS will manufacture a data set name for each of these two statements.
- The third `DD` statement has `DSN=&&WK2` coded. The `&&` tells z/OS to create a temporary data set.

`STEP2` references two of the three temporary data sets from `STEP1`.

- Since the data set name was explicitly coded (`DSN=&&WK2`) on the third `DD` statement in `STEP1`, the data set can be referenced by explicitly recoding the name on the first `DD` statement in `STEP2`.
- Since the data set name was omitted on the other `DD` statements of `STEP1`, the backward reference must be used on the second `DD` statement of `STEP2` to refer to any other data set name in `STEP1`.

Special DD statement parameters: DUMMY



© Copyright IBM Corporation 2011

Figure 3-23. Special DD statement parameters: DUMMY

ES074.0

Notes:

Code the positional parameter `DUMMY` on the `DD` statement to specify that:

- No device or external storage space is to be allocated to the data set.
- No disposition processing is to be performed on the data set.
- No input or output operations are to be performed on the data set.

If other parameters like `DISP`, `UNIT`, `SPACE`, and so on are coded on a `DD` statement with `DUMMY`, they are checked for SYNTAX but ignored.

DUMMY examples:

```
//OUTPUT    DD    DUMMY,DSN=A.B.C,DISP=(,CATLG),
//          SPACE=(TRK,(10,5)),UNIT=SYSDA
```

Special DD names



© Copyright IBM Corporation 2011

Figure 3-24. Special DD names

ES074.0

Notes:

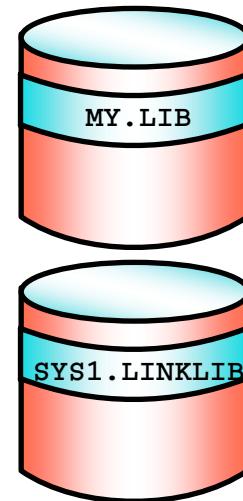
Use special DD names to specify such things as private libraries and data sets for storage dumps and checkpoints.

Code the special ddnames only when you want the special data sets.

//JOBLIB	Specifies a library that is searched for programs.
//JOBCAT	Is used to specify private catalogs.
//STEPLIB	Specifies a library that is searched for programs.
//STEPSAT	Is used to specify private catalogs.
//SYSUDUMP	Requests a SYSUDUMP dump if an ABEND occurs.
//SYSABEND	Requests a SYSABEND dump if an ABEND occurs.
//SYSMDUMP	Requests a SYSMDUMP dump if an ABEND occurs.
//SYSCHK	Identifies a data set used by checkpoint or restart.

Special DD statements: JOBLIB

```
//ZEBRA      JOB    66 ,TESTER,CLASS=T
//JOBLIB     DD     DSN=MY.LIB,DISP=SHR
//ONE        EXEC   PGM=FIRST
      .....
//TWO        EXEC   PGM=SECOND
      .....
//THREE      EXEC   PGM=THIRD
```



© Copyright IBM Corporation 2011

Figure 3-25. Special DD statements: JOBLIB

ES074.0

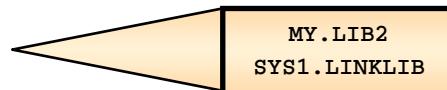
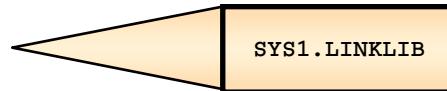
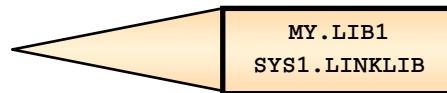
Notes:

A JOBLIB DD statement placed just after a JOB statement specifies a library that should be searched first for the programs executed by this job.

- Code the //JOBLIB DD statement to identify a private library/libraries the system is to search for each program in the job.
- The //JOBLIB DD statement alters the normal program search strategy.
- Code the //JOBLIB DD statement before the first EXEC statement.
- You are permitted to concatenate libraries to a JOBLIB DD statement. If you do so, the libraries are searched in their concatenation order.

Special DD statements: STEPLIB

```
//LIBJOB JOB      8877,TESTEM,CLASS=T
//ONE      EXEC   PGM=FIRST
//STEPLIB DD     DSN=MY.LIB1,DISP=SHR
.....
.....
.....
//TWO      EXEC   PGM=SECOND
.....
.....
.....
.....
.....
//THREE    EXEC   PGM=THIRD
//STEPLIB DD     DSN=MY.LIB2,DISP=SHR
```



© Copyright IBM Corporation 2011

Figure 3-26. Special DD statements: STEPLIB

ES074.0

Notes:

A STEPLIB DD statement placed just after an EXEC statement specifies a library that should be searched first for the program executed by the EXEC statement. A STEPLIB overrides a JOBLIB if both are used.

- //STEPLIB DD statement alters the normal program search strategy.
- When included in a step, the library named on the //STEPLIB DD statement is searched before the system searches the default library, SYS1.LINKLIB.
- You are permitted to concatenate libraries to a STEPLIB DD statement. If you do so, the libraries are searched in their concatenation order.



Important

Important rule to remember:

STEPLIB overrides JOBLIB in the step which has the STEPLIB statement.

To create a cataloged data set

UNIT= DSN=

DISP= VOL=

SPACE=



© Copyright IBM Corporation 2011

Figure 3-27. To create a cataloged data set

ES074.0

Notes:

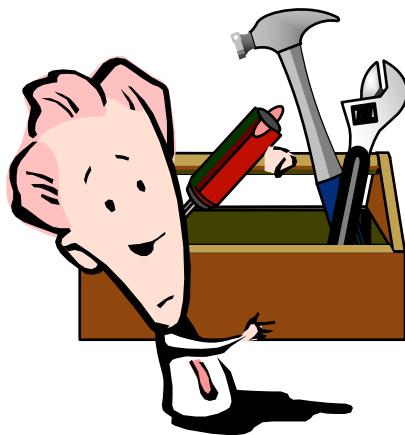
Complete the following DD statement parameters to create a new, cataloged data set.

```
//DDNAME DD  
//
```

To create a noncataloged data set

DSN=
UNIT=

DISP= VOL=
SPACE=



© Copyright IBM Corporation 2011

Figure 3-28. To create a noncataloged data set

ES074.0

Notes:

Complete the following DD statement parameters to create a new, noncataloged data set.

```
//DDNAME DD  
//
```

To reference an existing cataloged data set

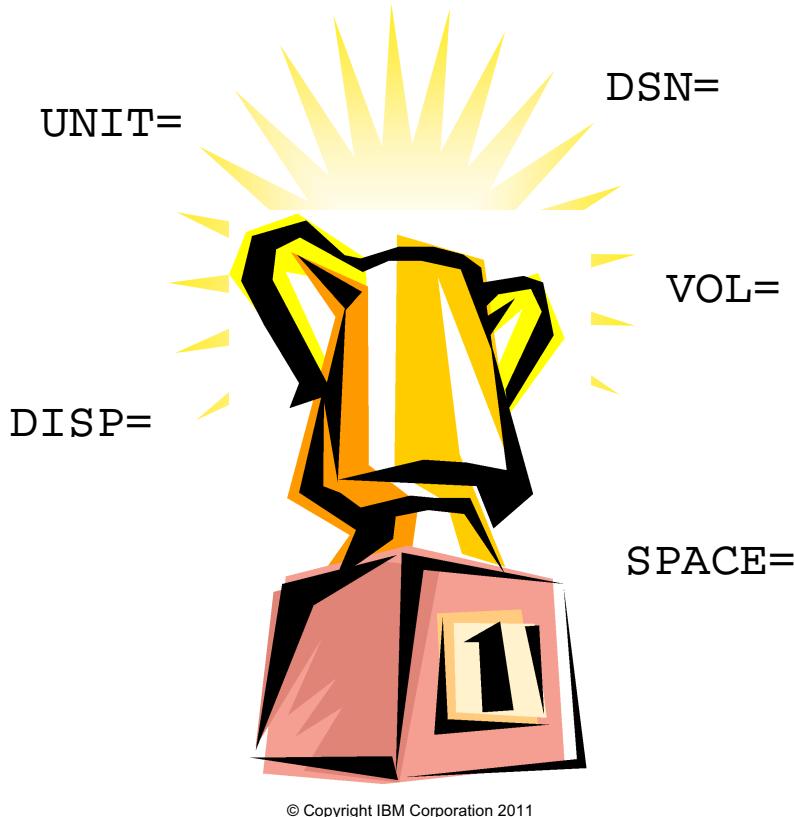


Figure 3-29. To reference an existing cataloged data set

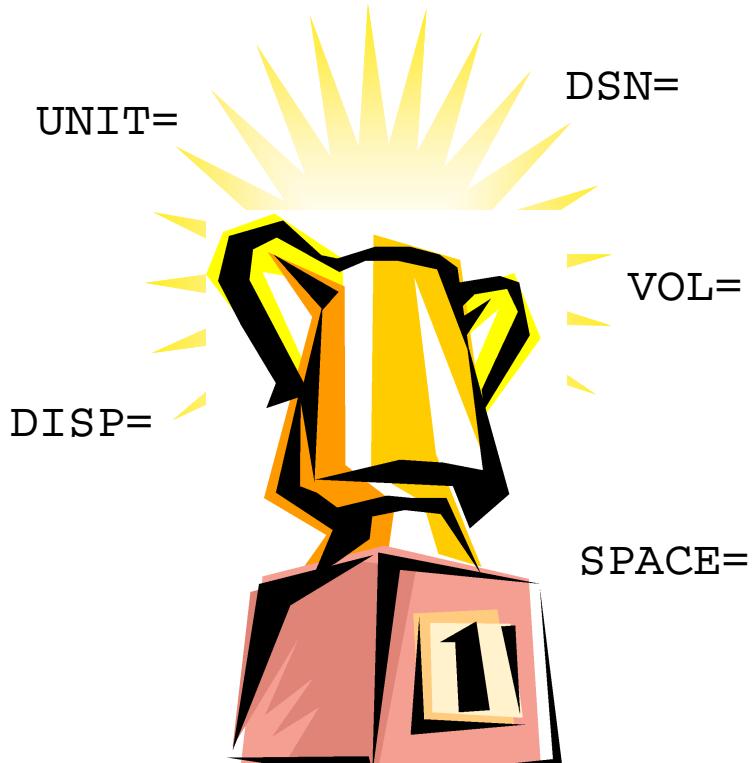
ES074.0

Notes:

Complete the following DD statement parameters to reference an existing cataloged data set.

```
//DDNAME DD  
//
```

To reference an existing noncataloged data set



© Copyright IBM Corporation 2011

Figure 3-30. To reference an existing noncataloged data set

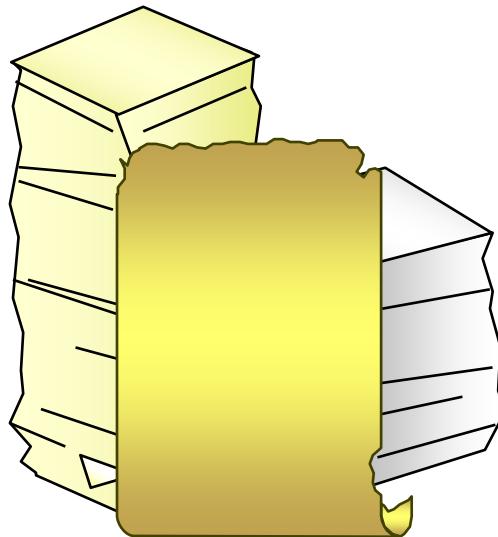
ES074.0

Notes:

Complete the following DD statement parameters to reference an existing noncataloged data set.

```
//DDNAME DD  
//
```

Job logs



© Copyright IBM Corporation 2011

Figure 3-31. Job logs

ES074.0

Notes:

Classroom exercise:

The following pages contain:

- A JCL stream
- A job log of a first run
- A job log of a second run
- A job log of a third run

Find the errors in the job logs from the first and second run.

Job stream

```
//TSOISBHC JOB 3MACHE3BMVS56W,HENDERS,CLASS=E,MSGCLASS=Q,
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K
//*****
//STEP01 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=*
//DD1      DD UMIT=SYSDA,VOL=SER=EDPAK5,DISP=OLD
//SYSIN  DD *
      SCRATCH DSNAME=TSOISBH.STEP1.PDS,VOL=SYSDA=EDPAK5
      UNCATLG DSNAME=TSOISBH.STEP1.PDS
//*****
//STEP02 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1  DD DSN=TSOISBH.BJCLLAB.TEST(SAMPLE),DISP=SHR
//SYSUT2  DD DSN=TSOISBH.STEP1.PDS(NEWDATA),DISP=(,KEEP),
//          UNIT=SYSDA,VOL=SER=EDPAK5
//          SPACE=(TRK,(1,1,2))
//SYSIN  DD DUMMY
//*****
//STEP03 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1  DD DSN=TSOISBH.STEP1.PDS(NEWDATA,DISP=SHR
//SYSUT2  DD SYSOUT=*,BLKSIZE=133
//SYSIN  DD DUMMY
```

© Copyright IBM Corporation 2011

Figure 3-32. Job stream

ES074.0

Notes:

Output from first run

```

J E S 2   J O B   L O G   --   S Y S T E M   M V S 1   --   N O D E   E S S M V S 1

10.18.10 JOB07731 ---- SATURDAY, 13 JUN 1998 ----
10.18.10 JOB07731 IRR010I USERID TSOISBH IS ASSIGNED TO THIS JOB.
10.18.10 JOB07731 IEFC452I TSOISBHC - JOB NOT RUN - JCL ERROR
----- JES2 JOB STATISTICS -----
      24 CARDS READ
      37 SYSOUT PRINT RECORDS
        0 SYSOUT PUNCH RECORDS
        2 SYSOUT SPOOL KBYTES
      0.00 MINUTES EXECUTION TIME
1  //TSOISBHC JOB 3MACHE3BMVS56W,HENDERS,CLASS=E,MSGCLASS=Q,          JOB07731
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K          00020000
//*****STEP01 EXEC PGM=IEHPROGM          00040000
2 //STEP01 EXEC PGM=IEBGENER          00095105
3 //SYSPRINT DD SYSOUT=*
4 //DD1 DD UNIT=SYSDA,VOL=SER=EDPAK5,DISP=OLD
5 //SYSIN DD *
//*****STEP02 EXEC PGM=IEBGENER          00095105
7 //SYSPRINT DD SYSOUT=*
8 //SYSUT1 DD DSN=TSOISBH.BJCLLAB.TEST(SAMPLE),DISP=SHR
9 //SYSUT2 DD DSN=TSOISBH.STEP1.PDS(NEWDATA),DISP=(,KEEP),
// UNIT=SYSDA,VOL=SER=EDPAK5
10 // SPACE=(TRK,(1,1,2))
11 //SYSIN DD DUMMY
//*****STEP03 EXEC PGM=IEBGENER          00095105
12 //STEP03 EXEC PGM=IEBGENER          00095105
13 //SYSPRINT DD SYSOUT=*
14 //SYSUT1 DD DSN=TSOISBH.STEP1.PDS(NEWDATA,DISP=SHR
15 //SYSUT2 DD SYSOUT=*,BLKSIZE=133
16 //SYSIN DD DUMMY
STMT NO. MESSAGE
      4 IEFC630I UNIDENTIFIED KEYWORD UUNIT
      10 IEFC605I UNIDENTIFIED OPERATION FIELD
      14 IEFC622I UNBALANCED PARENTHESES ON THE DD STATEMENT
© Copyright IBM Corporation 2011

```

Figure 3-33. Output from first run

ES074.0

Notes:

Output from second run (1 of 3)

```

J E S 2   J O B   L O G   --   S Y S T E M   M V S 1   --   N O D E   E S S M V S 1

10.23.03 JOB07732 ---- SATURDAY, 13 JUN 1998 -----
10.23.03 JOB07732 IRR010I USERID TSOISBH IS ASSIGNED TO THIS JOB.
10.23.05 JOB07732 ICH70001I TSOISBH LAST ACCESS AT 10:22:29 ON SATURDAY, JUNE 13, 1998
10.23.05 JOB07732 #HASP373 TSOISBHC STARTED - INIT 3 - CLASS E - SYS MVS1
10.23.05 JOB07732 IEF403I TSOISBHC - STARTED - TIME=10.23.05
10.23.06 JOB07732 IEF453I TSOISBHC - JOB FAILED - JCL ERROR - TIME=10.23.06
10.23.06 JOB07732 #HASP395 TSOISBHC ENDED
----- JES2 JOB STATISTICS -----
13 JUN 1998 JOB EXECUTION DATE
    24 CARDS READ
    81 SYSOUT PRINT RECORDS
    0 SYSOUT PUNCH RECORDS
    4 SYSOUT SPOOL KBYTES
    0.02 MINUTES EXECUTION TIME
1 //TSOISBHC JOB 3MACHE3BMVS56W,HENDERS,CLASS=E,MSGCLASS=A,          JOB07732
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K          00020000
//*****STEP01 EXEC PGM=IEHPROGM          00040000
2 //SYSPRINT DD SYSOUT=*
3 //DD1      DD UNIT=SYSDA,VOL=SER=EDPAK5,DISP=OLD
5 //SYSIN  DD *
//*****STEP02 EXEC PGM=IEBGENER          00095105
7 //SYSPRINT DD SYSOUT=*
8 //SYSUT1  DD DSN=TSOISBH.BJCLLAB.TEST(SAMPLE),DISP=SHR
9 //SYSUT2  DD DSN=TSOISBH.STEP1.PDS(NEWDATA),DISP=(,KEEP),
//           UNIT=SYSDA,VOL=SER=EDPAK5,
//           SPACE=(TRK,(1,1,2))
10 //SYSIN DD DUMMY
//*****STEP03 EXEC PGM=IEBGENER          00095105
11 //SYSPRINT DD SYSOUT=*
12 //SYSUT1  DD DSN=TSOISBH.STEP1.PDS(NEWDATA),DISP=SHR
14 //SYSUT2  DD SYSOUT=*,BLKSIZE=133
15 //SYSIN DD DUMMY

```

© Copyright IBM Corporation 2011

Figure 3-34. Output from second run (1 of 3)

ES074.0

Notes:

Output from second run (2 of 3)

```

ICH70001I TSOISBH LAST ACCESS AT 10:22:29 ON SATURDAY, JUNE 13, 1998
IEF236I ALLOC. FOR TSOISBHC STEP01
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO DDI
IEF237I JES2 ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP01 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   TSOISBH.TSOISBHC.JOB07732.D0000102.?           SYSOUT
IEF285I   SYS98164.T102305.RA000.TSOISBHC.R0006512        KEPT
IEF285I   VOL SER NOS= EDPAK5.
IEF285I   TSOISBH.TSOISBHC.JOB07732.D0000101.?           SYSIN
IEF373I STEP/STEP01 /START 98164.1023
IEF374I STEP/STEP01 /STOP 98164.1023 CPU    0MIN 00.03SEC SRB  0MIN 00.00SEC VIRT   48K SYS   244K EXT   4K SYS   9124K
IEF236I ALLOC. FOR TSOISBHC STEP02
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 229 ALLOCATED TO SYSUT1
IEGD100I 749 ALLOCATED TO DDNAME SYSUT2  DATACLAS (      )
IEF237I DMY ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP02 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   TSOISBH.TSOISBHC.JOB07732.D0000103.?           SYSOUT
IEF285I   TSOISBH.BJCLLAB.TEST                          KEPT
IEF285I   VOL SER NOS= EDPAK3.
IEF285I   TSOISBH.STEP1.PDS                           KEPT
IEF285I   VOL SER NOS= EDPAK5.
IEF373I STEP/STEP02 /START 98164.1023
IEF374I STEP/STEP02 /STOP 98164.1023 CPU    0MIN 00.02SEC SRB  0MIN 00.00SEC VIRT   140K SYS   196K EXT   4K SYS   9108K
IEF212I TSOISBHC STEP03 SYSUT1 - DATA SET NOT FOUND
IEF272I TSOISBHC STEP03 - STEP WAS NOT EXECUTED.
IEF373I STEP/STEP03 /START 98164.1023
IEF374I STEP/STEP03 /STOP 98164.1023 CPU    0MIN 00.00SEC SRB  0MIN 00.00SEC VIRT   OK SYS     OK EXT     OK SYS     OK
IEF375I JOB/TSOISBHC/START 98164.1023
IEF376I JOB/TSOISBHC/STOP 98164.1023 CPU    0MIN 00.05SEC SRB  0MIN 00.00SEC

```

© Copyright IBM Corporation 2011

Figure 3-35. Output from second run (2 of 3)

ES074.0

Notes:

Output from second run (3 of 3)

```
SYSTEM SUPPORT UTILITIES ---- IEHPROGM PAGE 0001

SCRATCH DSNAME=TSOISBH.STEP1.PDS, VOL=SYSDA=EDPAKS
NORMAL END OF TASK RETURNED FROM SCRATCH

UNCATLG DSNAME=TSOISBH.STEP1.PDS
NORMAL END OF TASK RETURNED FROM UNCATLG

UTILITY END
DATA SET UTILITY - GENERATE PAGE 0001
IEB352I WARNING : OUTPUT RECFM/LRECL COPIED FROM INPUT

PROCESSING ENDED AT EOD
```

© Copyright IBM Corporation 2011

Figure 3-36. Output from second run (3 of 3)

ES074.0

Notes:

Output from third run (1 of 3)

```

J E S 2 J O B L O G -- S Y S T E M M V S 1 -- N O D E E S S M V S 1

10.58.07 JOB07733 ---- SATURDAY, 13 JUN 1998 ----
10.58.07 JOB07733 IRR010I USERID TSOISBH IS ASSIGNED TO THIS JOB.
10.58.09 JOB07733 ICH70001I TSOISBH LAST ACCESS AT 10:56:21 ON SATURDAY, JUNE 13, 1998
10.58.09 JOB07733 *HASP373 TSOISBHC STARTED - INIT 1 - CLASS E - SYS MVS1
10.58.09 JOB07733 IEF403I TSOISBHC - STARTED - TIME=10.58.09
10.58.10 JOB07733 IEF404I TSOISBHC - ENDED - TIME=10.58.10
10.58.10 JOB07733 *HASP395 TSOISBHC ENDED

----- JES2 JOB STATISTICS -----
13 JUN 1998 JOB EXECUTION DATE
    24 CARDS READ
    117 SYSOUT PRINT RECORDS
        0 SYSOUT PUNCH RECORDS
        7 SYSOUT SPOOL KBYTES
    0.02 MINUTES EXECUTION TIME
1 //TSOISBHC JOB 3MACHE3BMV56W,HENDERS,CLASS=E,MSGCLASS=A,           JOB07733
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K                         00020000
//*****                                                 *****
2 //STEP01 EXEC PGM=IEHPROGM                                         00040000
3 //SYSPRINT DD SYSOUT=*
4 //DD1     DD UNIT=SYSDA,VOL=SER=EDPAK5,DISP=OLD
5 //SYSIN DD *
//*****                                                 *****
6 //STEP02 EXEC PGM=IEBGENER                                         00095105
7 //SYSPRINT DD SYSOUT=*
8 //SYSUT1 DD DSN=TSOISBH.BJCLLAB.TEST(SAMPLE),DISP=SHR
9 //SYSUT2 DD DSN=TSOISBH.STEP1.PDS(NEWDATA),DISP=(,CATLG),
//          UNIT=SYSDA,VOL=SER=EDPAK5,
//          SPACE=(TRK,(1,1,2))
10 //SYSIN DD DUMMY
//*****                                                 *****
11 //STEP03 EXEC PGM=IEBGENER                                         00095105
12 //SYSPRINT DD SYSOUT=*
13 //SYSUT1 DD DSN=TSOISBH.STEP1.PDS(NEWDATA),DISP=SHR
14 //SYSUT2 DD SYSOUT=*,BLKSIZE=133
15 //SYSIN DD DUMMY

```

© Copyright IBM Corporation 2011

Figure 3-37. Output from third run (1 of 3)

ES074.0

Notes:

Output from third run (2 of 3)

```

ICH70001I TSOISBH LAST ACCESS AT 10:56:21 ON SATURDAY, JUNE 13, 1998
IEF236I ALLOC. FOR TSOISBHC STEP01
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO DDI1
IEF237I JES2 ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP01 - STEP WAS EXECUTED - COND CODE 0000
IEF285I TSOISBH.TSOISBHC.JOB07733.D0000102.?           SYSOUT
IEF285I SYS98164.T105809.RA000.TSOISBHC.R0006511       KEPT
IEF285I VOL SER NOS= EDPAK5.
IEF285I TSOISBH.TSOISBHC.JOB07733.D0000101.?           SYSIN
IEF373I STEP/STEP01 /START 98164.1058
IEF374I STEP/STEP01 /STOP 98164.1058 CPU    0MIN 00.03SEC SRB   0MIN 00.00SEC VIRT   48K SYS   244K EXT   4K SYS   9132K
IEF236I ALLOC. FOR TSOISBHC STEP02
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 229 ALLOCATED TO SYSUT1
IGD100I 749 ALLOCATED TO DDNAME SYSUT2  DATACLAS (      )
IEF237I DMY ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP02 - STEP WAS EXECUTED - COND CODE 0000
IEF285I TSOISBH.TSOISBHC.JOB07733.D0000103.?           SYSOUT
IEF285I TSOISBH.BJCLLAB.TEST                         KEPT
IEF285I VOL SER NOS= EDPAK3.
IEF285I TSOISBH.STEP1.PDS                            CATALOGED
IEF285I VOL SER NOS= EDPAK5.
IEF373I STEP/STEP02 /START 98164.1058
IEF374I STEP/STEP02 /STOP 98164.1058 CPU    0MIN 00.02SEC SRB   0MIN 00.00SEC VIRT   140K SYS   196K EXT   4K SYS   9116K
IEF236I ALLOC. FOR TSOISBHC STEP03
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO SYSUT1
IEF237I JES2 ALLOCATED TO SYSUT2
IEF237I DMY ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP03 - STEP WAS EXECUTED - COND CODE 0000
IEF285I TSOISBH.TSOISBHC.JOB07733.D0000104.?           SYSOUT
IEF285I TSOISBH.STEP1.PDS                         KEPT
IEF285I VOL SER NOS= EDPAK5.
IEF285I TSOISBH.TSOISBHC.JOB07733.D0000105.?           SYSOUT
IEF373I STEP/STEP03 /START 98164.1058
IEF374I STEP/STEP03 /STOP 98164.1058 CPU    0MIN 00.02SEC SRB   0MIN 00.00SEC VIRT   96K SYS   192K EXT   4K SYS   9108K
IEF375I JOB/TSOISBHC/START 98164.1058
IEF376I JOB/TSOISBHC/STOP 98164.1058 CPU    0MIN 00.07SEC SRB   0MIN 00.00SEC

```

© Copyright IBM Corporation 2011

Figure 3-38. Output from third run (2 of 3)

ES074.0

Notes:

Output from third run (3 of 3)

```
SYSTEM SUPPORT UTILITIES ---- IEHPROGM PAGE 0001

SCRATCH DSNAME=TSOISBH.STEP1.PDS,VOL=SYSDA=EDPAK5
NORMAL END OF TASK RETURNED FROM SCRATCH

UNCATLG DSNAME=TSOISBH.STEP1.PDS
NORMAL END OF TASK RETURNED FROM UNCATLG

UTILITY END PAGE 0001
DATA SET UTILITY - GENERATE
IEB352I WARNING : OUTPUT RECFM/LRECL COPIED FROM INPUT

PROCESSING ENDED AT EOD PAGE 0001
DATA SET UTILITY - GENERATE

PROCESSING ENDED AT EOD
```

© Copyright IBM Corporation 2011

Figure 3-39. Output from third run (3 of 3)

ES074.0

Notes:

Printed output from step 3

```
*****
*****
**
*****
**
**
**
**
**
T H I S      I S      T H E
**
**
P R I N T E D      O U T P U T
**
**
F R O M      T H E      M E M B E R
**
**
S A M P L E
**
**
*****
**
*****
*****
```

© Copyright IBM Corporation 2011

Figure 3-40. Printed output from step 3

ES074.0

Notes:

Checkpoint (1 of 3)

1. What is the maximum length for a data set name?
 - a. 24
 - b. 40
 - c. 44
 - d. 48

2. True or False: UNIT=SYSDA is the same as UNIT=SYSALLDA.

3. What does UNIT=/3390 mean?
 - a. Allocate on unit number 3390
 - b. Allocate on any 3390-type DASD

4. Which of the following is a valid statement?
 - a. VOL=REF=DS3.JCL.CNTL
 - b. VOL=SER=123456
 - c. Both are valid statements.

© Copyright IBM Corporation 2011

Figure 3-41. Checkpoint (1 of 3)

ES074.0

Notes:

Checkpoint (2 of 3)

5. Code the SPACE parameter to allocate 20 cylinders in primary and 10 in secondary allocation.
6. In SPACE= (4096, (300,100, 50)), what is the meaning of 50?
 - a. 50 blocks of 4096 for secondary allocation
 - b. 50 blocks of directory allocation
7. What does DISP=OLD mean?
 - a. Data set exists
 - b. Job requires exclusive access to data set
 - c. The system starts writing at the end of the data set
 - d. Both data set exists and job requires exclusive access to data set
8. True or False: If DISP=MOD is coded for existing data set, records are appended to the logical end of data.

© Copyright IBM Corporation 2011

Figure 3-42. Checkpoint (2 of 3)

ES074.0

Notes:

Checkpoint (3 of 3)

9. Name two ways by which the system can locate private libraries for program execution.

10. What is the meaning of DISP= (NEW, CATLG, DELETE) ?

11. Which of the following are valid temporary data set names?
 - a. DSN=&TEMP
 - b. DSN=&&TEMP
 - c. DSN=

© Copyright IBM Corporation 2011

Figure 3-43. Checkpoint (3 of 3)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Create a new data set
- Reference an existing uncataloged data set
- Reference an existing cataloged data set
- Code and discuss the DISP parameter
- Use special DD statements
- Detect and correct JCL syntax and usage errors

© Copyright IBM Corporation 2011

Figure 3-44. Unit summary

ES074.0

Notes:

Unit 4. Introduction to utilities and conditional execution

What this unit is about

Certain routine tasks involving data set organization and maintenance arise frequently in a data processing environment. It would take substantial time to repeatedly write programs to handle these tasks. Therefore, many IBM-supplied utilities are available to process these commonly occurring tasks.

Situations also arise when it might be desirable to skip certain steps in a job. For example, suppose the second step of a two-step job processes output from step one. If step one fails to produce the output for step two, it would probably be pointless attempting to execute step two. This unit introduces utility routines and discusses the bypassing of job steps.

What you should be able to do

After completing this unit, you should be able to:

- Identify the different types of utility programs
- Establish the rules for coding utility control statements
- Describe the functions of system and data set utility programs
- Review the JCL and utility control statements appropriate for:
 - IEBGENER
 - IEBPTPCH
 - IEHLIST
 - IDCAMS
- Use the *DFSMS/dfp Utilities* manual for reference
- Discuss the COND parameter and conditional execution of job steps

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597	<i>z/OS MVS JCL Reference</i>
SC26-7414	<i>DFSMS/dfp Utilities</i>

SC26-7394

*DFSMS/dfp Access Method Services for the
Integrated Catalog Facility*

Unit objectives

After completing this unit, you should be able to:

- Identify the different types of utility programs
- Establish the rules for coding utility control statements
- Describe the functions of system and data set utility programs
- Review the JCL and utility control statements appropriate for:
 - IEBGENER
 - IEBPTPCH
 - IEHLIST
 - IDCAMS
- Use the *DFSMS/dfp Utilities* manual for reference
- Discuss the COND parameter and conditional execution of job steps

© Copyright IBM Corporation 2011

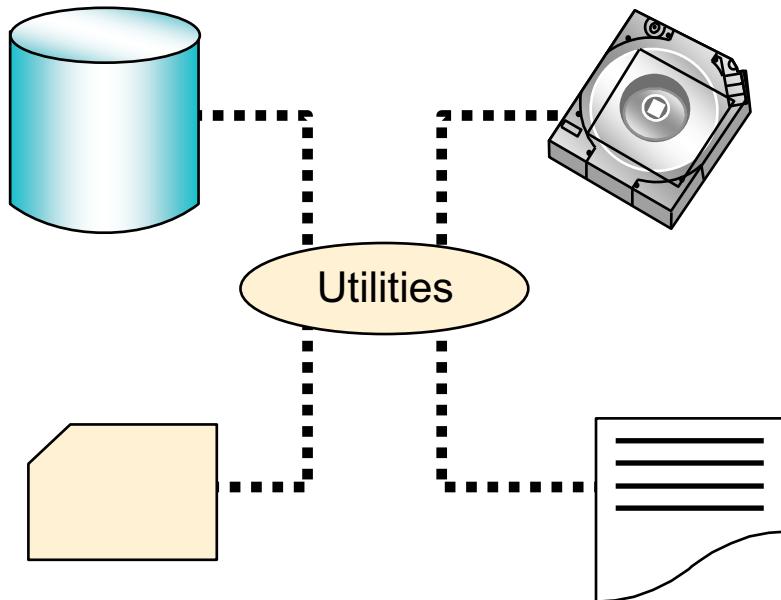
Figure 4-1. Unit objectives

ES074.0

Notes:

z/OS utilities

Programs that organize and maintain data



© Copyright IBM Corporation 2011

Figure 4-2. z/OS utilities

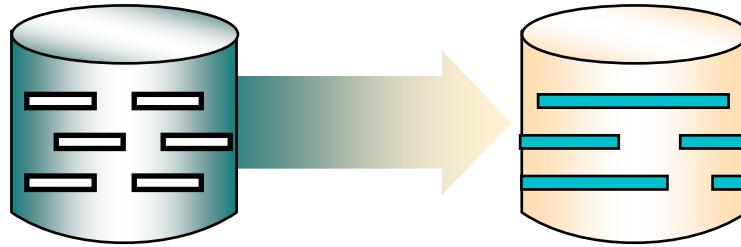
ES074.0

Notes:

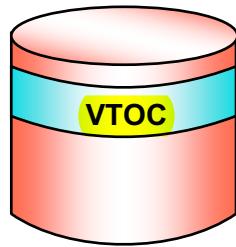
- Utilities are IBM-supplied programs which perform certain routine and frequently occurring tasks in the z/OS environment.
- Utilities are used in DASD, tape, print, and punch operations.
- Utilities are also used to list the contents of a VTOC and allocate, update, delete, catalog, and uncatalog data sets.
- Many of the tasks performed by the batch utilities described in the utilities manual can also be performed under Interactive System Productivity Facility /Program Development Facility (ISPF/PDF). It is sometimes preferable to use the batch utilities and vice versa.
- Advantages of ISPF/PDF over the batch utilities:
 - Speed in performing a few utilitarian tasks
 - Ease of use
- Advantages of the batch utilities over ISPF/PDF:
 - Preparing tasks for automation
 - Performing a task between steps in a batch job
 - Printed documentation

Classes of utilities

Data set: IEB _____ (Data set/record oriented)



System: IEH _____ (Volume oriented)



© Copyright IBM Corporation 2011

Figure 4-3. Classes of utilities

ES074.0

Notes:

The two most important classes of utilities are data set and system.

- Data set utilities (prefixed with IEB) are used to:
 - Copy, print, punch, update, reorganize, and compare data at the data set or record level or both.
- System utilities (prefixed with IEH) are used to:
 - List VTOC information
 - Copy, delete, catalog, and uncatalog data sets
 - Write tape labels and to add or delete data set passwords
- Both sets of utilities are controlled by *JCL statements* and *utility control statements*.

Utility selection

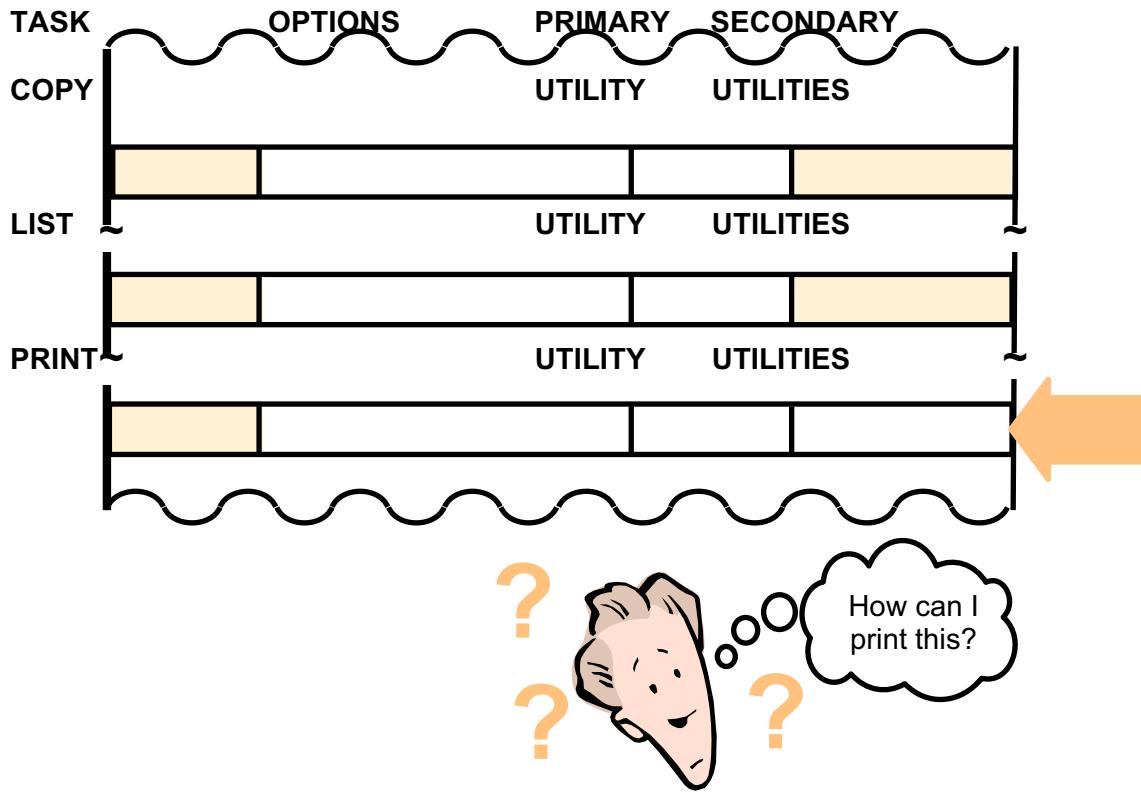


Figure 4-4. Utility selection

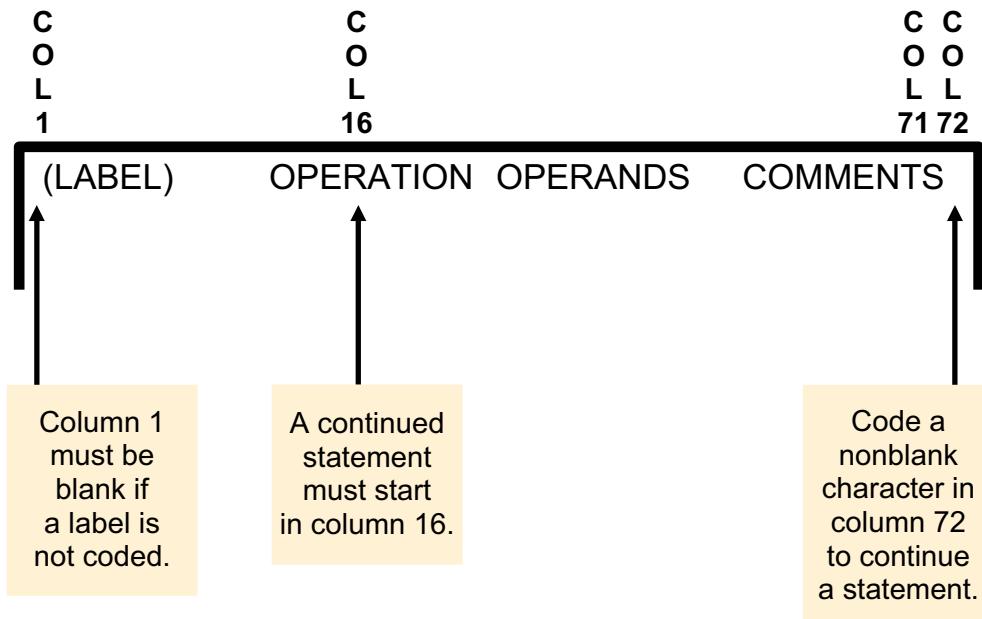
ES074.0

Notes:

The introductory chapter in the utilities manual contains a section, Guide to Utility Program Functions, which can be used to select the appropriate utility for a given task.

- TASK column shows tasks you might want to perform.
- OPTIONS column shows a more specific definition of the task.
- PRIMARY UTILITY column identifies the utility especially suited for the task.
- SECONDARY UTILITIES column identifies other utilities which might be used.

Utility control statement format



© Copyright IBM Corporation 2011

Figure 4-5. Utility control statement format

ES074.0

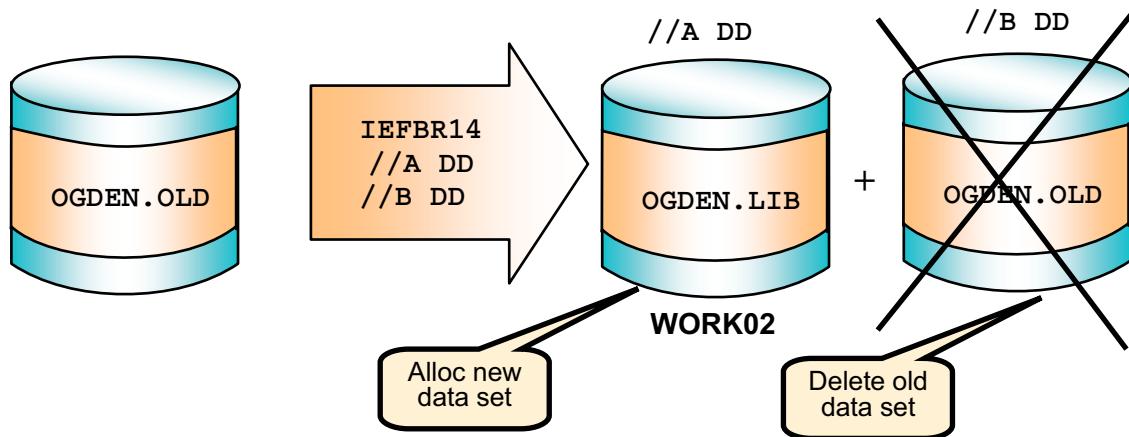
Notes:

Utility control statements:

- Identify a function to be performed.
- Do not begin with // in columns 1 and 2.*
- Contain four fields:
 - Label field:** Optional
 - Operation field:** Required
 - Operands field:** Required
 - Comments field:** Optional
- Must be coded in columns 1 to 71.
- If the statement exceeds column 71, use the continuation rules:
 - Break the statement in column 71 or after a comma.
 - Code a non-blank character in column 72.
 - Continue the statement in column 16 of the following line.

IEFBR14 allocation

another name is dummy



```
//STEP1 EXEC PGM=IEFBR14
//A DD DSN=OGDEN.LIB.CNTL,DISP=(NEW,CATLG),VOL=SER=WORK02,
// UNIT=3390,SPACE=(CYL,(3,1,25)
//B DD DSN=OGDEN.OLD.DATA,DISP=(OLD,DELETE)
```

© Copyright IBM Corporation 2011

Figure 4-6. IEFBR14 allocation

ES074.0

Notes:

The only function of the above program is to provide a zero (0) completion code. It is used as a safe vehicle to execute JCL.

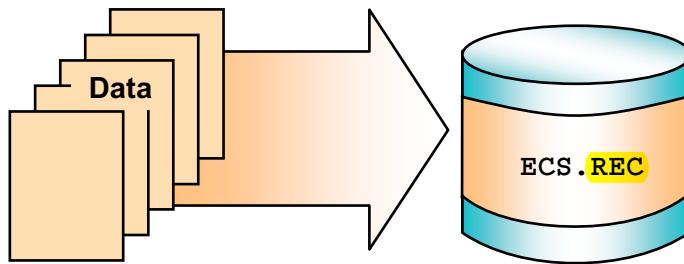
The notion of executing JCL is considered incorrect terminology, but it conveys the idea very well. For example, consider the above job.

This is a useful job, although the program that is executed (IEFBR14) does nothing.

While preparing to run the job, the initiator allocates OGDEN.LIB.CNTL and keeps the data set when the job ends. It also deletes OGDEN.OLD.DATA at the end of the job. The DD names A and B have no meaning and are used because the syntax of a DD statement requires a DD name.

The same functions to create one data set and delete another could be done through ISPF, for example, but these actions might be needed as part of a larger sequence of batch jobs.

IEBGENER card-to-disk



```
//BUILD      EXEC    PGM=IEBGENER
//SYSPRINT   DD SYSOUT=*
//SYSUT1     DD          *
      JONES    FRED    53AF 87 5701 NINE MILE ROAD
      ANDERSON  DON     78AF 34 320 WESTHEIMER #219
//SYSUT2     DD DSN=ECS.REC,DISP=(,CATLG,DELETE),
//              SPACE=(TRK,(1,1)),VOL=SER=WORK01,
//              RECFM=FB,LRECL=80,UNIT=SYSDA
//SYSIN      DD DUMMY
```

© Copyright IBM Corporation 2011

Figure 4-7. IEBGENER card-to-disk

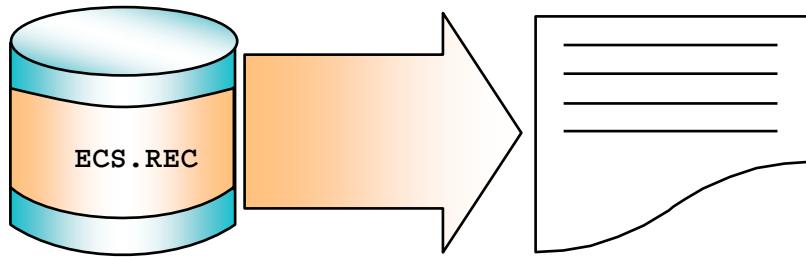
ES074.0

Notes:

IEBGENER is the most basic copy or list program supplied with z/OS. It has been present since the first release of OS/360.

- IEBGENER is a data set utility used to create, copy, or print sequential data sets.
- The example allocates a data set, ECS.REC, and then copies instream data to the data set.
 - The EXEC statement specifies the program to be executed.
 - The SYSPRINT DD statement defines the message data set.
 - The SYSUT1 DD statement defines the input data set.
 - The SYSUT2 DD statement defines the output data set.
 - The SYSIN DD statement defines the control data set. This is where IEBGENER looks for utility control statements. If none are required, as in this example, //SYSIN DD DUMMY can be coded. IEBGENER is performing its default function of copying input to output with no editing required.

IEBGENER copy/print



```
//PRINT      EXEC   PGM=IEBGENER
//SYSPRINT   DD     SYSOUT=*
//SYSUT1     DD     DSN=ECS.REC,DISP=SHR
//SYSUT2     DD     SYSOUT=*
//SYSIN      DD     *
               GENERATE MAXFLDS=3
               RECORD FIELD=(10,20,,1),FIELD=(10,1,,15),FIELD=(6,50,,30)
```

© Copyright IBM Corporation 2011

Figure 4-8. IEBGENER copy/print

ES074.0

Notes:

IEBGENER is used for copying or printing all or selected portions of data sets.

This step executes IEBGENER to create the SYSUT2 output data in a different form from the input data set ECS.REC.

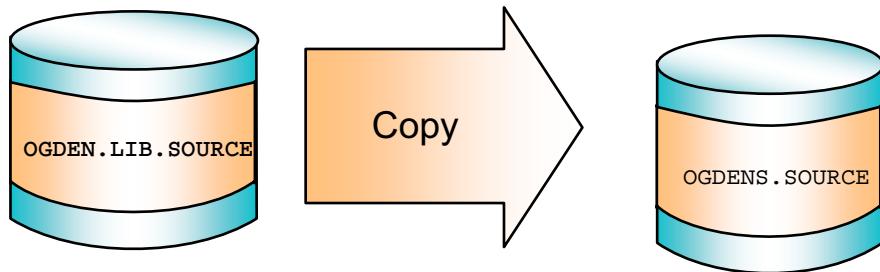
The SYSIN DD * statement indicates that instream records, or control statements, follow.

The GENERATE statement specifies that editing is to be performed and the operand MAXFLDS=3 indicates that no more than three fields will be described on the subsequent RECORD statement.

The RECORD statement describes the input and output fields through the FIELD parameter. The format is:

FIELD=(LENGTH OF FIELD, POSITION IN INPUT, CONVERSION, POSITION IN OUTPUT)

IEBCOPY copy



```
//COPY EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DISP=SHR,DSN=OGDEN.LIB.SOURCE
//SYSUT2 DD DISP=(NEW,KEEP),UNIT=TAPE,DSN=OGDENS.SOURCE,
// VOL=SER=123456
```

© Copyright IBM Corporation 2011

Figure 4-9. IEBCOPY copy

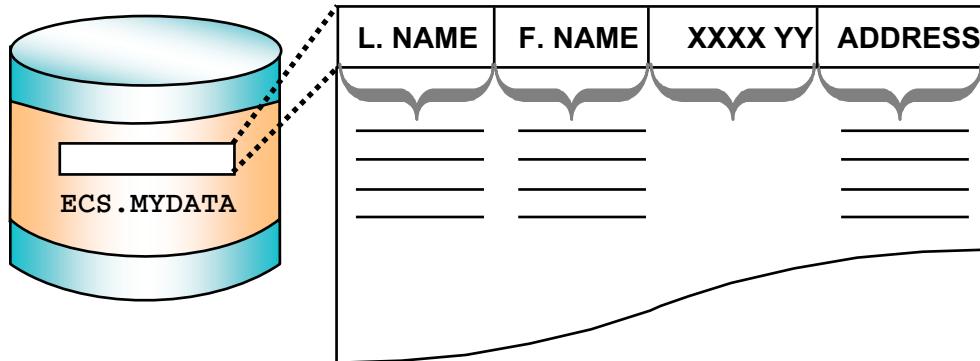
ES074.0

Notes:

This utility is commonly used for several purposes:

- To copy selected (or all) members from one partitioned data set to another.
- To copy a partitioned data set into a unique sequential format known as an unloaded partitioned data set. As a sequential data set, it can be written on tape, sent by FTP, or manipulated as a simple sequential data set.
- To read an unloaded partitioned data set (which is a sequential file) and recreate the original partitioned data set. Optionally, only selected members might be used.
- To compress partitioned data sets (in place) to recover lost space.

IEBPTPCH print



```
//PRINT      EXEC   PGM=IEBPTPCH
//SYSPRINT   DD     SYSOUT=*
//SYSUT1     DD     DSN=ECS . MYDATA,DISP=SHR
//SYSUT2     DD     SYSOUT=*
//SYSIN      DD     *
                  PRINT TYPORG=PS,MAXFLDS=3
                  TITLE ITEM= ('EMPLOYEES OF ABC COMPANY',27)
                  TITLE ITEM= ('NAME'                   ADDRESS',15)
                  RECORD FIELD=(8,2,,10),FIELD=(5,10,,20),FIELD=(36,25,,31)
```

© Copyright IBM Corporation 2011

Figure 4-10. IEBPTPCH print

ES074.0

Notes:

IEBPTPCH is used to print or punch all or selected portions of data sets. Editing can be done on the data.

This step executes IEBPTPCH and reads records from SYSUT1 data set ECS.MYDATA and prints selected information from each record.

The EXEC statement specifies program to be executed.

The SYSPRINT DD statement defines the message data set.

The SYSUT1 DD statement defines the input data set.

The SYSUT2 DD statement defines the output data set.

The SYSIN DD * statement indicates that instream records, or control statements, follow.

The PRINT control statement specifies that the data set has an organization of *physical sequential* and that a maximum of three fields will be printed on output.

The output titles are specified on the TITLE control statements.

The RECORD control statement describes the input and output fields via the FIELD parameter. The format is:

FIELD=(*LENGTH OF FIELD*,*POSITION IN INPUT*,*CONVERSION*,*POSITION IN OUTPUT*)

IEBPTPCH print output

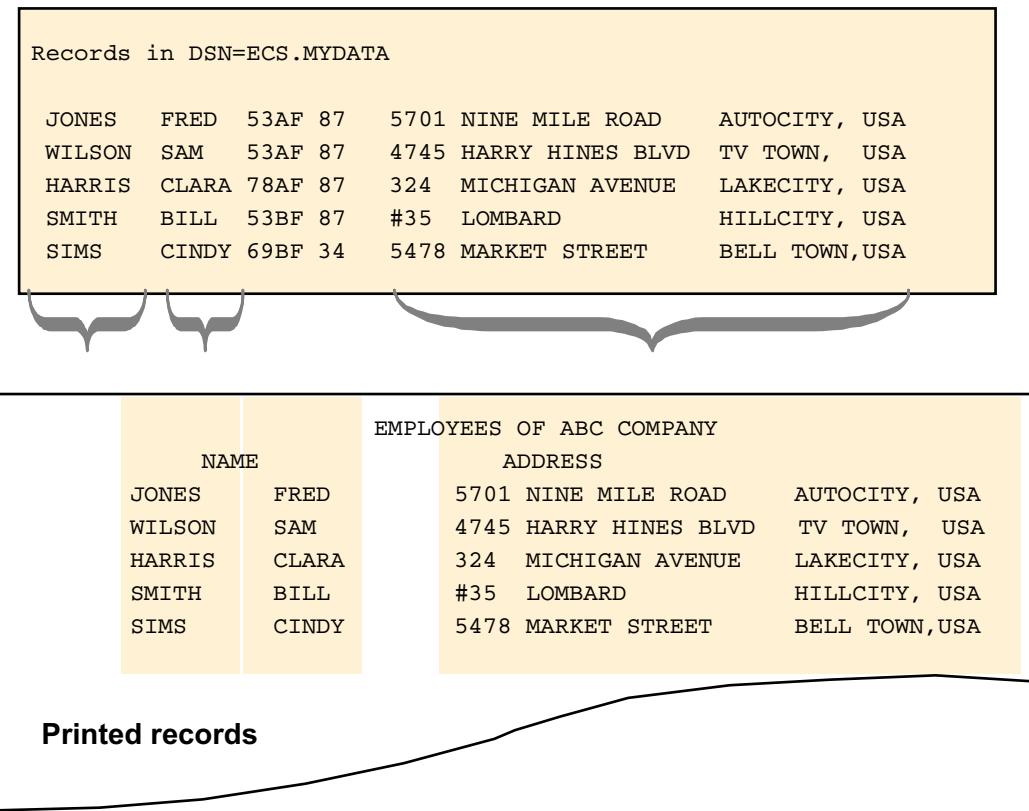


Figure 4-11. IEBPTPCH print output

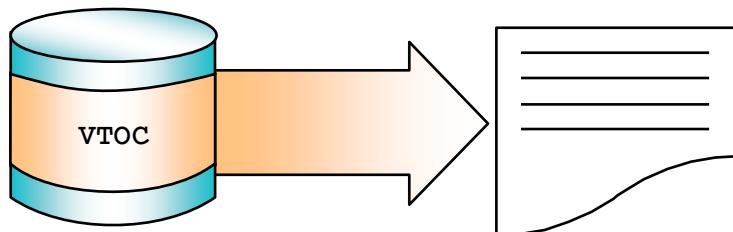
ES074.0

Notes:

The first part of the listing shows the five records as they exist in the input data set ECS MYDATA

The second part of the listing shows the reformatted and printed records from the data set. Two fields in the input records were not selected for printing.

IEHLIST LISTVTOC



```
//LISTVTOC EXEC PGM=IEHLIST
//SYSPRINT DD SYSOUT=*
//DD1 DD DISP=OLD,UNIT=SYSDA,VOL=SER=A1A1
//DD2 DD DISP=OLD,UNIT=SYSDA,VOL=SER=NEWCDB
//SYSIN DD *
LISTVTOC FORMAT,VOL=SYSDA=A1A1
LISTVTOC FORMAT,VOL=SYSDA=NEWCDB
DSNAME=(ESDB3.VM76N.LIST,ESDB3.MVS76N.LIST)
```

Col 72

© Copyright IBM Corporation 2011

Figure 4-12. IEHLIST LISTVTOC

ES074.0

Notes:

The **IEHLIST** utility is used to list a partitioned data set directory or a disk volume VTOC. It is normally used for VTOC listings and provides bit-level information.

- `IEHLIST` can be used to list entries in a DASD VTOC.
 - The `JOB` statement introduces the job/processing requirements.
 - The `EXEC` statement specifies the program to be executed.
 - The `SYSPRINT DD` statement defines the message data set.
 - The `DD1` and `DD2 DD` statements allocate packs needed for the `LISTVTOC` operation.
 - `SYSIN DD` statement defines the control data set. This is where `IEHLIST` looks for utility control statements.
 - The first `LISTVTOC` control statement requests an edited (FORMAT) VTOC listing for pack A1A1.
 - The second `LISTVTOC` control statement requests an edited (FORMAT) VTOC listing for two data sets, `ESDB3.VM76N.LIST` and `ESDB3.MVS76N.LIST`.

**Note**

DSNAME *cannot* be abbreviated as DSN on a control statement.

The VOL parameter format is:

VOL=XXXXX=YYYYYY

where XXXXX is the value specified for the UNIT parameter on the JCL DD statement and YYYYYYY is the SER subparameter value.

IEHLIST is not used often in most installations, but is needed in the rare cases where a VTOC is corrupted. It is sometimes used with the SUPERZAP program to patch or fix a broken VTOC.

LISTVOC output

SYSTEMS SUPPORT UTILITIES--IEHLIST

PAGE 1

DATE, 1990.338 TIME, 10.59.58
CONTENTS OF VTOC ON VOL A1A1 <THIS VOLUME IS NOT SMS MANAGED>
THERE IS A 1 LEVEL VTOC INDEX
DATA SETS ARE LISTED IN ALPHANUMERIC ORDER
FORMAT 4 DSCB NO AVAIL/MAX DSCB/MAX DIRECT NO AVAIL NEXT ALT FORMAT 6 LAST FMT 1 VTOC EXTENT THIS DSCB
V1 DSCBS PER TRK BLK PER TRK ALT TRK TRK(C-H) (C-H-R) DSCB(C-H-R)/LOW(C-H) HIGH(C-H) (C-H-R)
81 7152 53 46 15 1770 0 9 14 53 1 0 9 14 1 0 1
-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
SYS1.VTOCIX.A1A1 A1A1 1 1990.094 00.000 00.000 1 PS F 00 2048
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
2048 TRKS CONTIG 0 9 18 0 1 0 3
EXTENTS NO LOW(C-H) HIGH(C-H)
0 10 0 10 9
---ON THE ABOVE DATA SET, THERE ARE 0 EMPTY TRACK(S).
-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
SYS1.VVDS.VA1A1 A1A1 1 1990.138 00.000 00.000 1 VS * 80 4096
SMS.IND LRECL KEYLEN INITIAL ALOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
0 TRKS 10 0 0 47968 1 0 5
EXTENTS NO LOW(C-H) HIGH(C-H)
0 0 1 0 10
---UNABLE TO CALCULATE EMPTY SPACE.
VPSM A = NUMBER OF TRKS IN ADDITION TO FULL CYLS IN THE EXTENT
TRK FULL TRK FULL TRK FULL TRK FULL TRK FULL TRK FULL
ADDR CYLS A
11 0 4 160 1759 5
THERE ARE 1759 EMPTY CYLINDERS PLUS 9 EMPTY TRACKS ON THIS VOLUME
THERE ARE 7151 BLANK DSCBS IN THE VTOC ON THIS VOLUME
THERE ARE 175 UNALLOCATED VIRS IN THE INDEX

© Copyright IBM Corporation 2011

Figure 4-13. LISTVOC output

ES074.0

Notes:

This is a VTOC listing for only the pack A1A1.

The pack contains two data sets:

- SYS1.VTOCIX.A1A1 (indexed VTOC)
 - Sequential
 - Unblocked
 - Fixed-length records
 - Created on the 94th day of 1990
 - 10 track data set in cylinder 10
- SYS1.VVDS.VA1A1 (VSAM volume data set VVDS)
 - Ten-track VSAM data set in cylinder 0
 - Created on the 138th day of 1990

The volume has available for allocation:

- 1759 cylinders
- Nine tracks

The VTOC contains 7151 blank DSCBs.

IDCAMS: DELETE

Cataloged non-VSAM data set

```
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD      SYSOUT=*
//SYSIN     DD  *
               DELETE -
               TSOUSR.PROD.DATA
```

© Copyright IBM Corporation 2011

Figure 4-14. IDCAMS: DELETE

ES074.0

Notes:

The IDCAMS program is not part of the basic set of z/OS utilities documented in the z/OS utilities manual. The IDCAMS program is primarily used to create and manipulate VSAM data sets. It has other functions (such as catalog updates), but it is most closely associated with VSAM. It provides many complex functions. The basic IBM manual describing these functions is *DFSMS Access Method Services for Catalogs*.

- IDCAMS can be used to delete non-VSAM data sets.
- This step executes IDCAMS to delete the non-VSAM data set, TSOUSR.PROD.DATA.
- The SYSIN DD * statement indicates that instream records, or control statements, follow.
- The DELETE statement specifies the name of the data set to be deleted.
- Rules for coding control statements for IDCAMS:
 - Column 1 must be blank.
 - A dash (-) or plus sign (+) can be used for continuation.
 - Multiple control statements can be coded.
- IDCAMS requires fewer DD statements and can be used for both VSAM and non-VSAM data sets.

Conditional execution of JOB steps

- COND parameter
 - Specifies when a step should *not* execute
 - Coded on JOB or EXEC statement
 - Supported in every version/release of z/OS

© Copyright IBM Corporation 2011

Figure 4-15. Conditional execution of JOB steps

ES074.0

Notes:

z/OS provides facilities to conditionally execute job steps depending on the outcome of previous job steps.

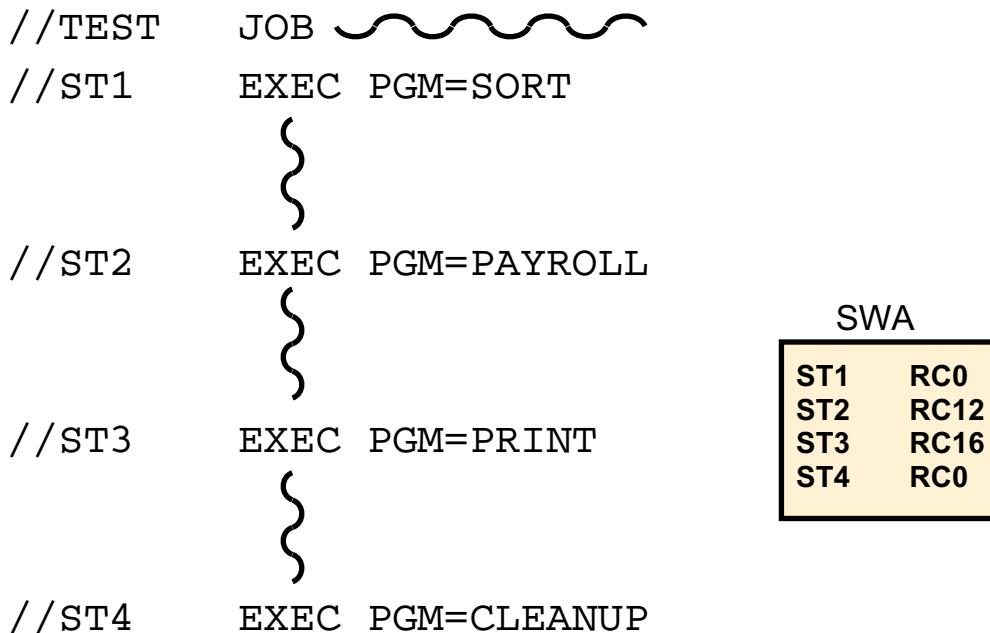
Two techniques are provided for testing the success or failure of job steps and determining whether to continue execution based on the outcome of the tests:

- COND parameter
- IF/THEN/ELSE/ENDIF statement construct

The COND is supported in every version/release of z/OS and MVS.

COND can be coded on the JOB or EXEC statement.

Return code setting



© Copyright IBM Corporation 2011

Figure 4-16. Return code setting

ES074.0

Notes:

Job steps can be selectively bypassed based on the return code of a preceding step or steps.

Programs assign a return code to signify a certain condition. If this condition occurs during execution, the program sets the return code.

The return code is not set if the step does not execute.

Return codes for every step in the job are stored for the life of the job.

Use the COND parameter on the JOB and EXEC statements to test return codes.

Most IBM programs produce standard return codes.

JOB return code

- New JOBRC parameter in JOB statement

```
JOBRC= {MAXRC}  
       {LASTRC}  
       { (STEP, stepname [.procstepname] ) }
```

© Copyright IBM Corporation 2011

Figure 4-17. JOB return code

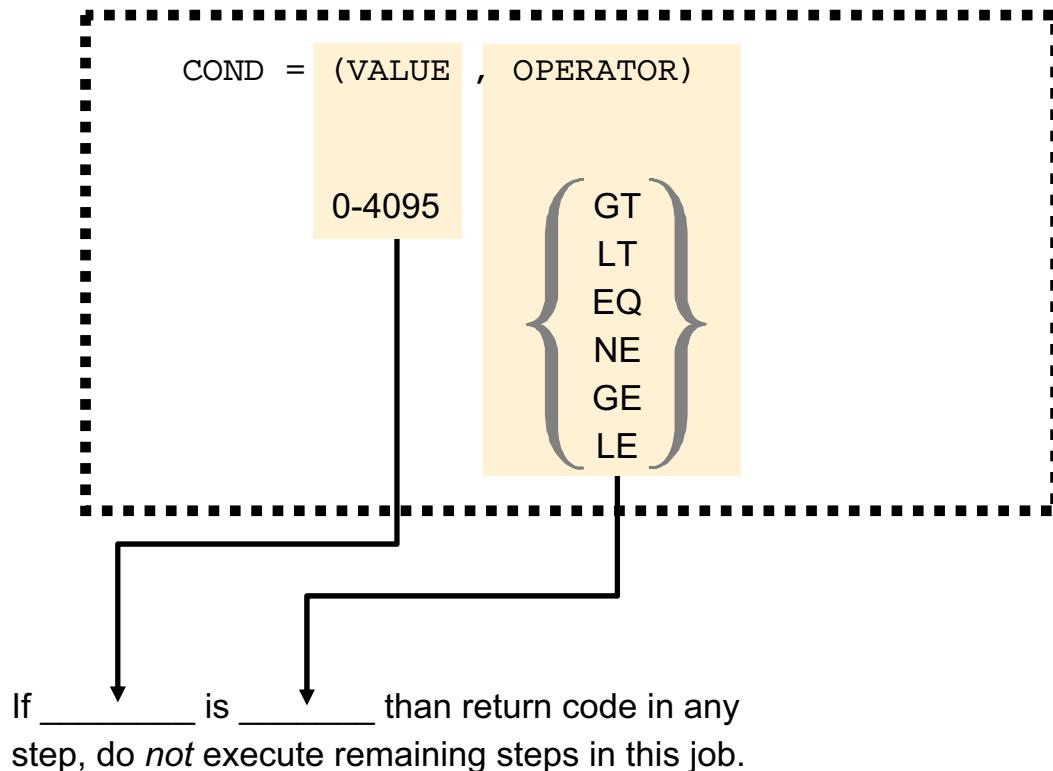
ES074.0

Notes:

New JOB subparameter JOBRC was introduced with z/OS V1R13 - JOBRC.

Use the JOBRC parameter to control how the job completion code (presented by JES2 or JES3) is set. By default (when JOBRC is not specified), the job completion code is set to the highest return code of any step, or if the job's execution fails because of an ABEND, the job completion code is set to the last ABEND code; however, this parameter can be used to request that the job completion code be set to the return code of the last executed step.

COND parameter on the JOB statement



© Copyright IBM Corporation 2011

Figure 4-18. COND parameter on the JOB statement

ES074.0

Notes:

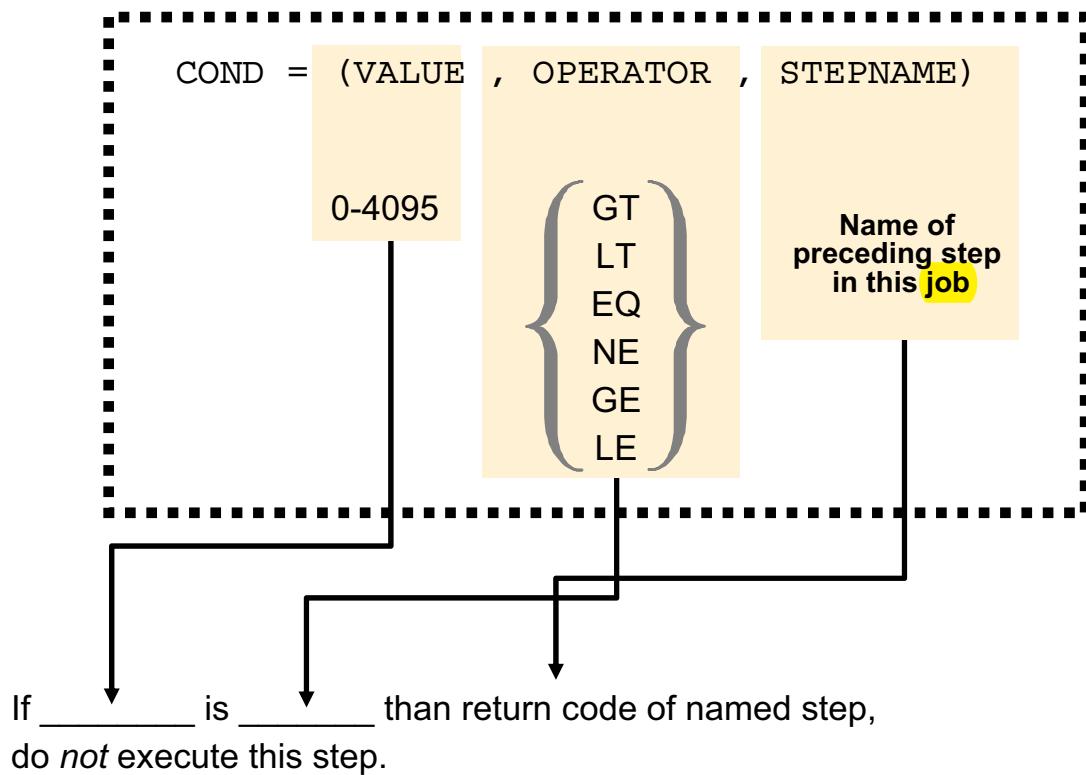
Return code tests are made for each step in the job *except* the first.

If any condition tested is true, bypass all remaining steps in the job.

A maximum of eight separate return code tests can be specified.

Return code tests specified on the JOB statement are performed before tests specified on an EXEC statement.

COND parameter on the EXEC statement



© Copyright IBM Corporation 2011

Figure 4-19. COND parameter on the EXEC statement

ES074.0

Notes:

The return code test is performed on the steps specified.

If no steps are specified, the test is performed on all preceding steps.

If any condition tested is true, bypass the testing job step.

A maximum of eight separate return code tests can be specified. This allows the testing of different return codes on different steps.

Abnormal termination testing

- COND=EVEN
 - Execute this step even if a prior step ABENDED.
- COND=ONLY
 - Execute this step only if a prior step ABENDED.

© Copyright IBM Corporation 2011

Figure 4-20. Abnormal termination testing

ES074.0

Notes:

Bypassing a step because of a return code test is not the same as abnormally terminating (ABENDING) the step. The system ABENDs a step following a serious error that prevents proper execution. Bypassing a step is simply omitting the step.

If a step ABENDs, the system bypasses all following steps in the job.

Code COND=EVEN on the EXEC statement to specify that this step is to execute whether or not some previous step ABENDs.

Code COND=ONLY on the EXEC statement to specify that this step is to execute only if some previous step ABENDs.

COND testing example

//ST1	EXEC PGM=IEBGENER	RC 0
	{	
//ST2	EXEC PGM=PAYROLL,COND=(8,LE,ST1)	RC 12
	{	
//ST3	EXEC PGM=PRINT,COND=((8,LE,ST1),(12,EQ,ST2))	NO RC
//ST4	EXEC PGM=CLEANUP,COND=((20,EQ),EVEN)	RC 0

© Copyright IBM Corporation 2011

Figure 4-21. COND testing example

ES074.0

Notes:

ST2 checks the return code of ST1:

COND=(8,LE,ST1)

If 8 is less than or equal 8, bypass the step. ST1 RC is 0, so the test is false, and ST2 executes.

ST3 tests the return code of both ST1 and ST2:

COND=((8,LE,ST1),(12,EQ,ST2))

If 8 is less than or equal the RC from ST, bypass the step. ST1 RC is 0, so the test is false, and testing continues. Second test checks if 12 is equal to RC from ST2. It is, test is true, so ST3 is bypassed.

ST4 tests for EVEN and a return code from all previous steps:

COND=((20,EQ),EVEN)

Execute this step even if a prior step ABENDED. No prior step ABENDED, so testing continues. If the RC from any previous step is equal to 20, bypass the step. If the test is false (no previous RCs are equal to 20), the step executes.

Checkpoint (1 of 2)

1. What is the name of the utility to manage VSAM data sets?
2. True or False: IEFBR14 can be used to delete an existing data set.
3. True or False: IEBGENER can manage both sequential and PDS data sets.
4. Which utility can you use to copy a PDS?
 - a. IEBCOPY
 - b. IEBGENER
 - c. IDCAMS

© Copyright IBM Corporation 2011

Figure 4-22. Checkpoint (1 of 2)

ES074.0

Notes:

Checkpoint (2 of 2)

5. With IEBGENER, what is the DD name for input data set?
 - a. SYSIN
 - b. SYSUT1
 - c. SYSUT2

6. Name two system utilities (IEH) and their use.

7. How can you code the COND parameter to force the system to execute a following step?

8. If a step ABENDs, the system bypasses:
 - a. All following steps in the job
 - b. The next step in the job

© Copyright IBM Corporation 2011

Figure 4-23. Checkpoint (2 of 2)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Identify the different types of utility programs
- Establish the rules for coding utility control statements
- Describe the functions of system and data set utility programs
- Review the JCL and utility control statements appropriate for:
 - IEBGENER
 - IEBPTPCH
 - IEHLIST
 - IDCAMS
- Use the *DFSMS/dfp Utilities* manual for reference
- Discuss the COND parameter and conditional execution of job steps

© Copyright IBM Corporation 2011

Figure 4-24. Unit summary

ES074.0

Notes:

Unit 5. Data management, organization, and format

What this unit is about

z/OS supports data sets with many different attributes and characteristics. For example, z/OS works with many data sets containing records with different lengths and formats. Some data sets contain executable programs, while others contain data to be processed by those programs. Some data sets are managed by the Storage Management Subsystem, but others are not. This unit looks at the different attributes and characteristics of data sets supported by z/OS.

What you should be able to do

After completing this unit, you should be able to:

- Examine the record formats supported by z/OS
- Compare blocked and unblocked data sets
- Discuss system-determined block size
- Introduce the SMS-managed data sets
- Describe the data set organizations supported by z/OS
- Differentiate between PDS and PDSE data sets
- Explain how to access an z/OS UNIX System Services HFS data set and files
- Discuss VSAM data set creation through JCL

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597	<i>z/OS MVS JCL Reference</i>
SA22-7598	<i>z/OS MVS JCL User's Guide</i>

Unit objectives

After completing this unit, you should be able to:

- Examine the record formats supported by z/OS
- Compare blocked and unblocked data sets
- Discuss system-determined block size
- Introduce the SMS-managed data sets
- Describe the data set organizations supported by z/OS
- Differentiate between PDS and PDSE data sets
- Explain how to access a z/OS UNIX System Services HFS data set and files
- Discuss VSAM data set creation through JCL

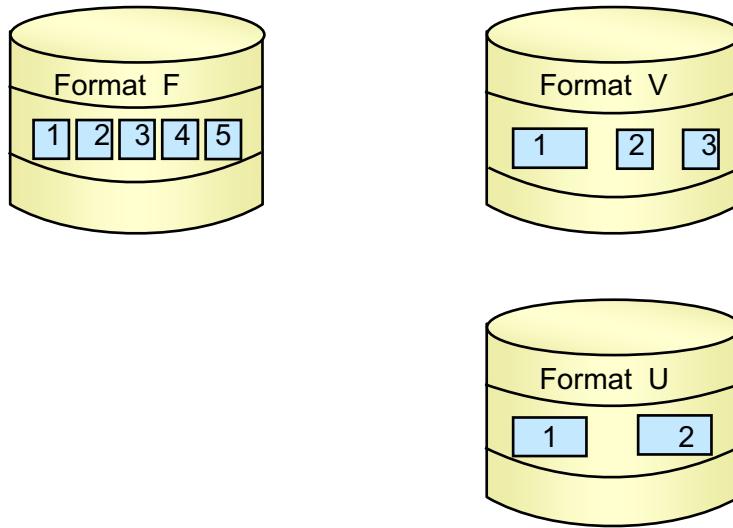
© Copyright IBM Corporation 2011

Figure 5-1. Unit objectives

ES074.0

Notes:

Record formats



© Copyright IBM Corporation 2011

Figure 5-2. Record formats

ES074.0

Notes:

Data set record formats

Traditional z/OS data sets are record-oriented. In normal usage, there are no byte-stream files such as are found in PC and UNIX systems. (z/OS UNIX has byte-stream files, and byte-stream functions exist in other specialized areas. These are not considered to be traditional data sets.) Records are either fixed-length or variable-length in a given data set. When editing a data set with ISPF, for example, each line is a record.

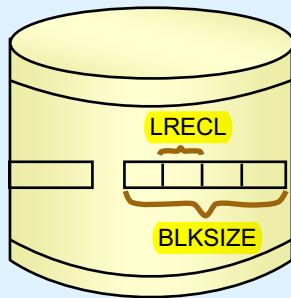
Record format (RECFM)

A record is the basic unit of information used by a program. Records are stored in three basic formats:

- **RECFM=F:** Fixed-length records (DASD and tape).
This means that all records in the data set have the same length.
- **RECFM=V:** Variable-length records (DASD and tape).
This implies that not all records in the data set have the same length.
- **RECFM=U:** Undefined-length records (DASD and tape).
Format U permits the processing of records that do not conform to either F or V format.

Data control block parameter

- Specifies characteristics of a data set:
 - **LRECL**: The record length
 - **RECFM**: The record format
 - **BLKSIZE**: The block length



There are over 30 more subparameters!

© Copyright IBM Corporation 2011

Figure 5-3. Data control block parameter

ES074.0

Notes:

DASD volumes are used for storing data and executable programs (including the operating system itself) and for temporary working storage. One DASD volume can be used for many different data sets, and space on it can be reallocated and reused.

The maximum length of a logical record (LRECL) is limited by the physical size of the media used.

We must stress the difference between a block and a record. A *block* is what is written on disk, while a *record* is a logical entity.

A *logical record* is a unit of information about a unit of processing (for example, a customer, an account, a payroll employee, and so on). It is the smallest amount of data to be processed, and it is comprised of fields that contain information recognized by the processing application.

Logical records, when located in DASD, tape, or optical devices, are grouped in physical records called *blocks*.

- The record format and two other parameters are especially important in describing records in a data set:
 - The logical record length (LRECL) describes the length of the records which the program creates or analyzes.
 - The block size (BLKSIZE) describes the length of the blocks which are written to DASD or tape. For fixed length records, BLKSIZE is a multiple of LRECL.
- z/OS allows *all* DCB keyword subparameters to be coded either as full DD statement parameters or as traditional DCB subparameters. For example, z/OS allows either coding below:

DCB=(LRECL=_____,RECFM=_____,BLKSIZE=_____)

or

LRECL=_____,RECFM=_____,BLKSIZE=_____

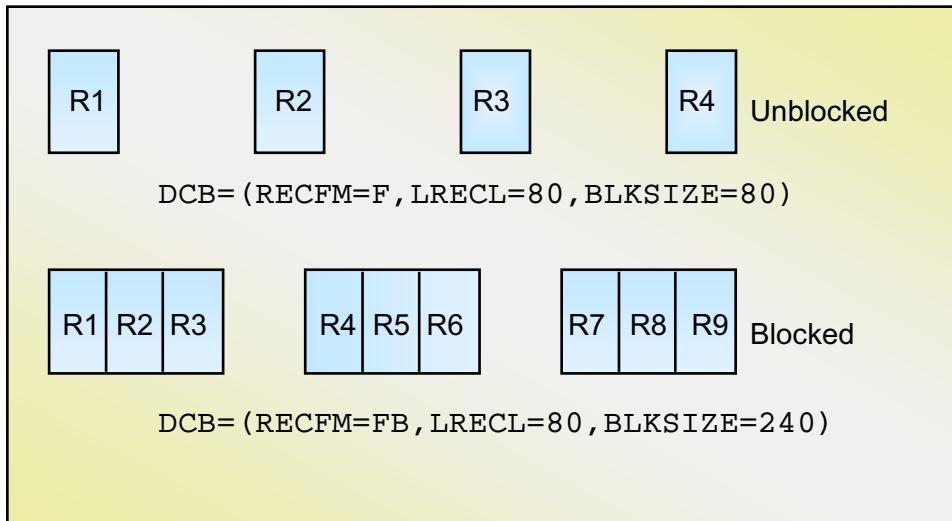
DCB examples:

```
//OUTDD    DD  DSN=ABC,DISP=(,CATLG),UNIT=SYSDA,
//          SPACE=(CYL,(5,2),RLSE),DCB=(LRECL=80,RECFM=FB,BLKSIZE=3200)
```

or

```
//OUTDD    DD  DSN=ABC,DISP=(,CATLG),UNIT=SYSDA,
//          SPACE=(CYL,(5,2),RLSE),LRECL=80,RECFM=FB
```

Fixed-length record format



© Copyright IBM Corporation 2011

Figure 5-4. Fixed-length record format

ES074.0

Notes:

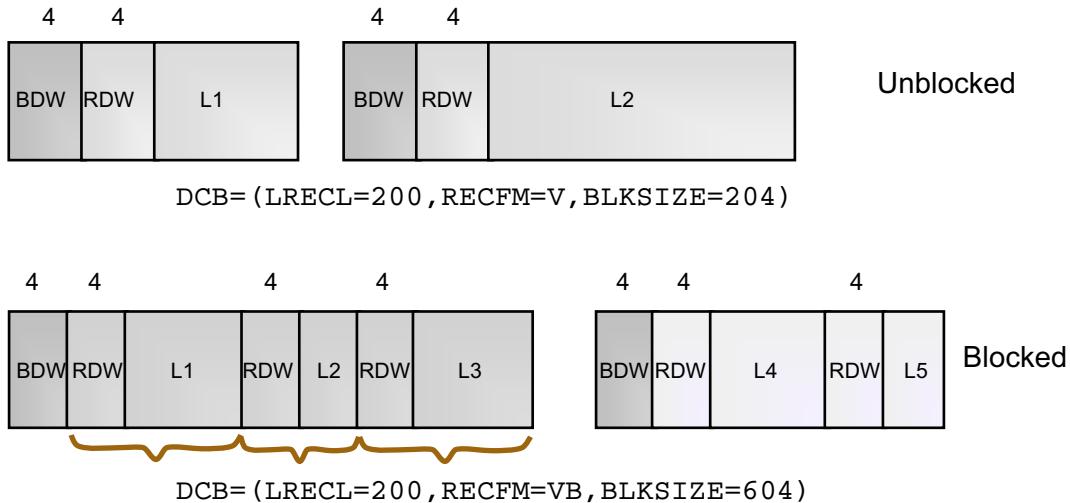
Example 1 `LRECL=80, BLKSIZE=80, RECFM=F`

Example 2 `LRECL=80, BLKSIZE=240, RECFM=FB`

Blocked data sets are usually recommended over unblocked data sets for at least two reasons:

- Less space is required to store data in a blocked data set than an unblocked data set.
- Less time is required to read or write the same amount of data to a blocked data set than an unblocked data set.

Variable-length record format



© Copyright IBM Corporation 2011

Figure 5-5. Variable-length record format

ES074.0

Notes:

If format-V records are unblocked, the block size must be 4 bytes greater than the record length (LRECL).

Control information describing the records and blocks is stored in:

- **Block Descriptor Word (BDW):** Built by the system and controls the number of bytes actually read or written.
- **Record Descriptor Word (RDW):** Built by the programmer and is used to deblock and/or to control the movement of data to programmer-defined work areas.

Variable-block spanned format

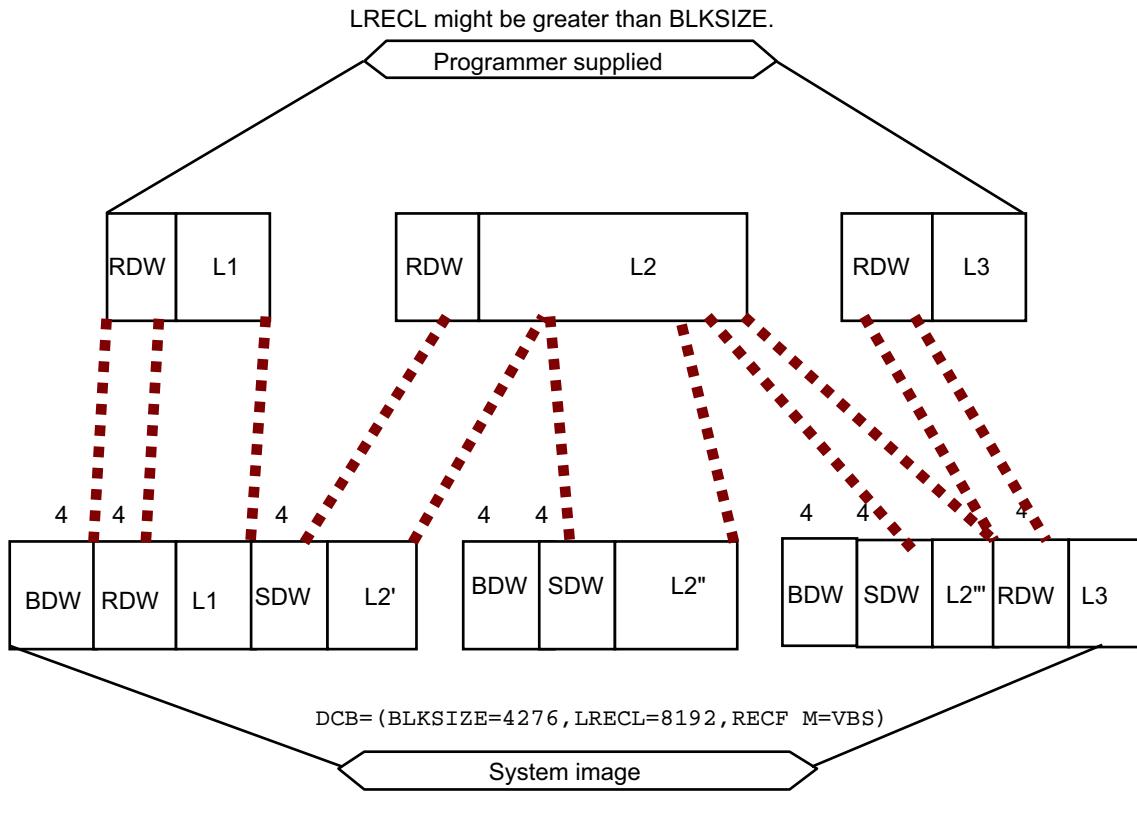


Figure 5-6. Variable block spanned format

ES074.0

Notes:

LRECL can exceed BLKSIZE. For example, the maximum BLKSIZE supported by z/OS is 32,760 bytes; however, logical records can be larger than this.

If LRECL exceeds BLKSIZE, the system will generate enough SDWs to fit the logical record into as many blocks as required. When the record is read, the system will reformat the segmented record so the program encounters only RDWs.

The format of the BDW and RDW is the same:

- Bytes 0 and 1: Length of the record or block
- Bytes 2 and 3: These reserved fields are set to binary zeros.

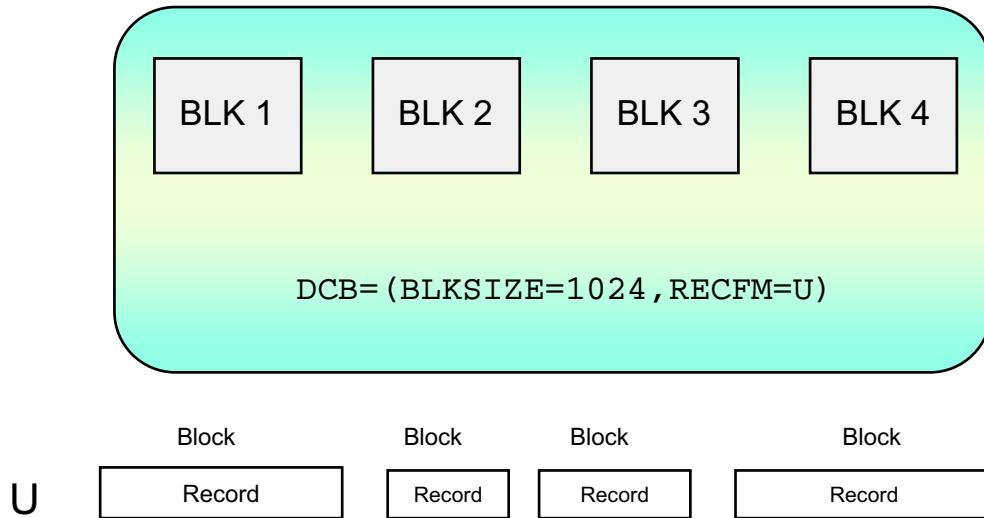
The format of the SDW is:

- Byte 0 and 1: Length of the segment
- Byte 2: Specifies if this is first, last, or interior segment
- Byte 3: This reserved field is set to binary zero.

RECFM=VS is similar to RECFM=VBS except that RECFM=VS does not allow two different records to occupy the same physical block.

Undefined-length record format

BLKSIZE 32,760



© Copyright IBM Corporation 2011

Figure 5-7. Undefined-length record format

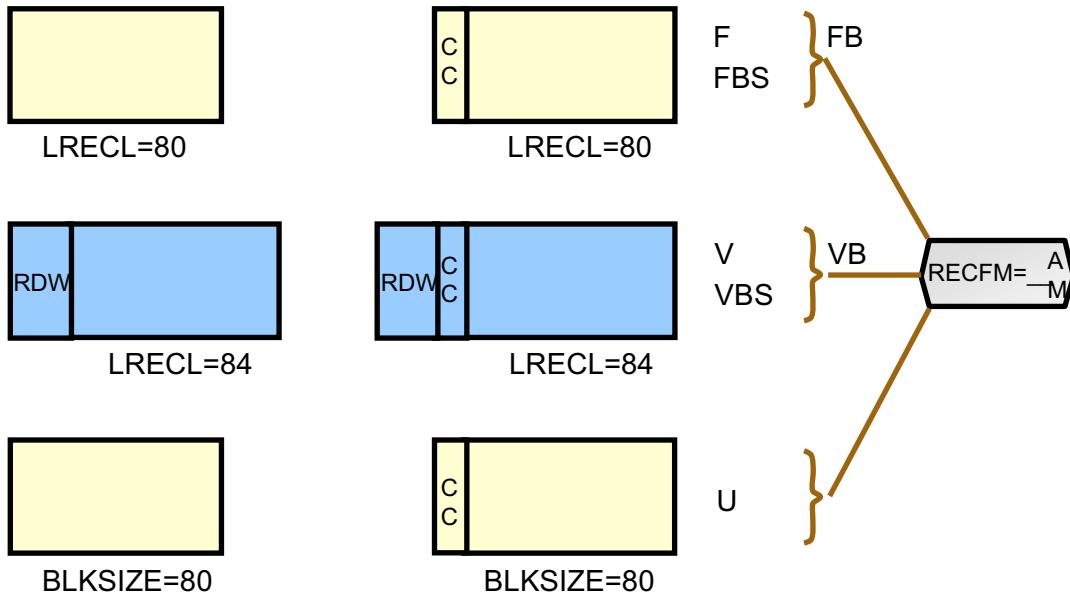
ES074.0

Notes:

If no record format is coded, undefined-length format is the default.

Undefined records have no defined internal structure for access method.

Control characters



© Copyright IBM Corporation 2011

Figure 5-8. Control characters

ES074.0

Notes:

Control characters are used to control printer spacing and special punch options like stacker reset.

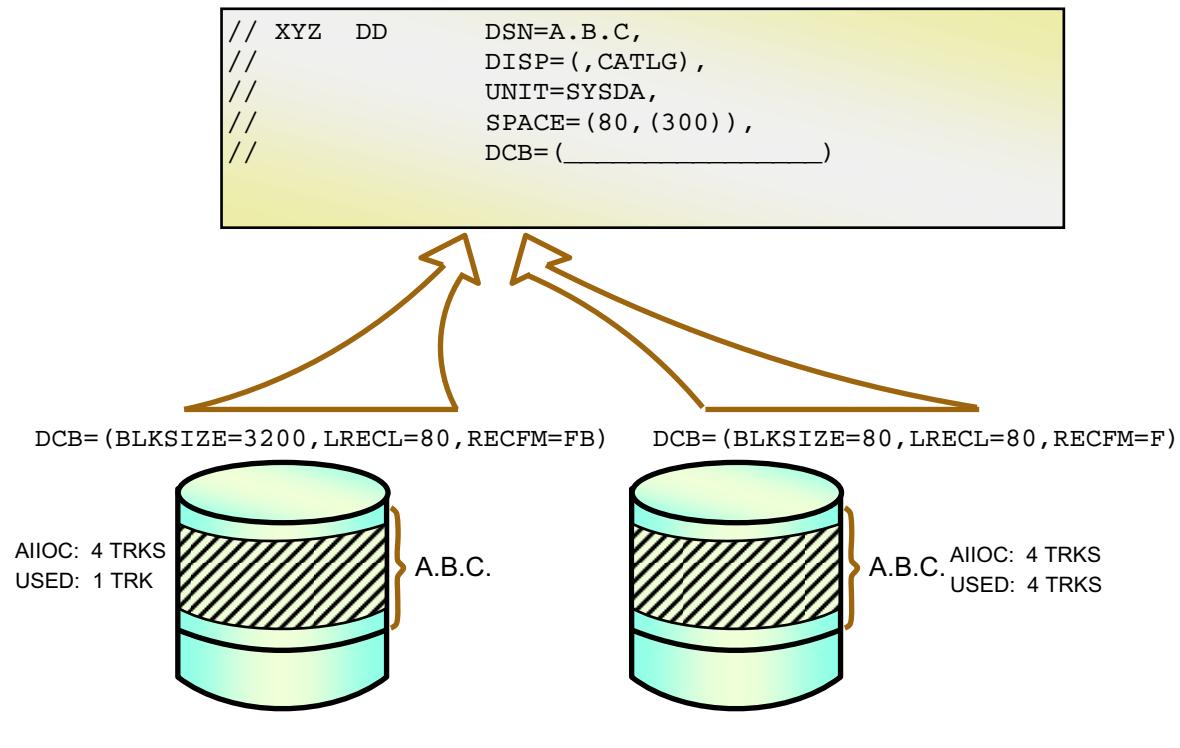
All control characters are 1 byte.

- **A:** ISO/ANSI control characters (space then print)
- **M:** Machine control characters (print then space)

Example:

RECFM=FBA

Space versus blocking



© Copyright IBM Corporation 2011

Figure 5-9. Space versus blocking

ES074.0

Notes:

The parameter `SPACE= (80, (300))` requests space for 300 80-byte records. The system will allocate four tracks on a 3390.

Now, suppose a program wants to write 300 80-byte records to a 3390.

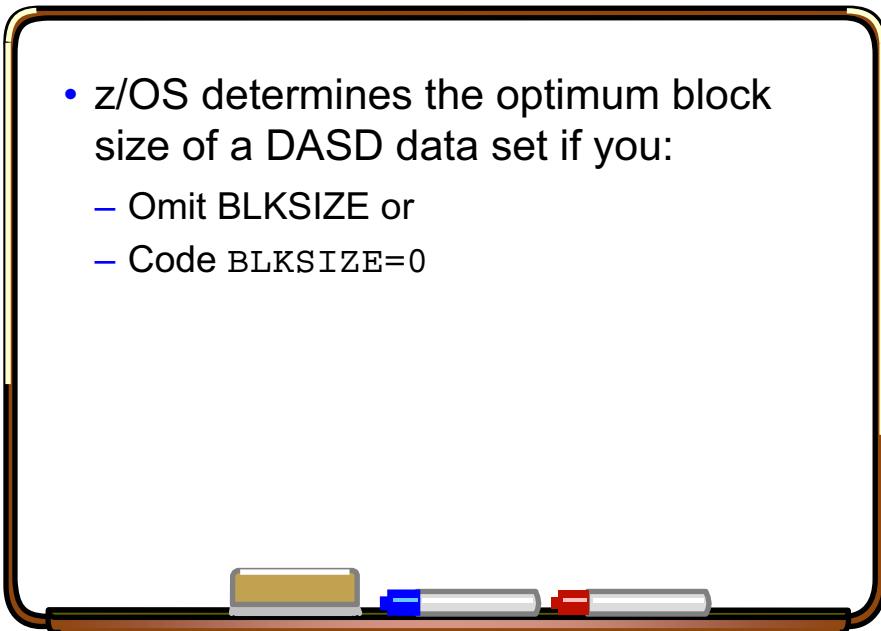
Example 1:

- If `DCB= (BLKSIZE=3200, LRECL=80, RECFM=FB)`, the system will place 40 logical records in each block. A 3390 device can hold 14 3200-byte blocks per track, so a 3390 track can hold a maximum of 560 logical records. (Note that $40 \times 14 = 560$.) Thus, less than one 3390 track can easily hold all 300 80-byte records.

Example 2:

- If `DCB= (BLKSIZE=80, LRECL=80, RECFM=F)`, the system will place one logical record in each block. A 3390 device can hold 78 80-byte blocks per track, so a 3390 track can hold only 78 logical records! Thus, almost four 3390 tracks are required to hold 300 80-byte records.

System-determined block size



© Copyright IBM Corporation 2011

Figure 5-10. System-determined block size

ES074.0

Notes:

System-determined block size

The system can derive the best block size for DASD, tape, and spooled data sets. The system does not derive a block size when the RECFM is U.

z/OS will calculate the optimum block size for a DASD data set if four conditions are true:

- RECFM=FB__ or RECFM=VB__ (RECFM requests a blocked data set)
- LRECL is coded
- The data set is sequential or partitioned (to be discussed later)
- BLKSIZE is omitted or specified to be zero

Benefits of letting z/OS determine BLKSIZE:

- More efficient use of DASD space
- Improved performance
- Device independence
- Better productivity

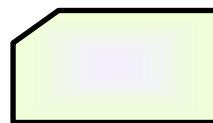
The DCB merge

DCB subparameters can be specified in the:

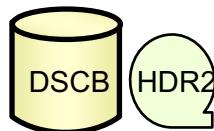
1. Program



2. JCL



3. VTOC or label



© Copyright IBM Corporation 2011

Figure 5-11. The DCB merge

ES074.0

Notes:

DCB subparameters can be specified in/on the:

- Program (These subparameters cannot be overridden)
- JCL DD statement
- VTOC or tape label

Since DCB subparameters can be specified in three different places, there is always a possibility of encountering confusion in what the system will use for some parameter. For example, someone might code three different BLKSIZE values: one value in the program, another value in the VTOC, and still another value in JCL. This situation can produce errors during the processing of the data. To avoid this possibility:

- Do not code DCB subparameters (for example, BLKSIZE) in the program.
- Code DCB subparameters (for example, BLKSIZE) during data set creation. Then, they will be stored in the VTOC or in the tape label (for standard label tapes).
- Do not code DCB subparameters (for example, BLKSIZE) in JCL for existing data sets.
- Furthermore, it is good practice not to code BLKSIZE at all. The system will determine it through system-determined blocksize.

Data set types supported

- Data set types supported:

- VSAM data sets
- Non-VSAM data sets
- Extended-format data sets
- Objects
- z/OS UNIX files

© Copyright IBM Corporation 2011

Figure 5-12. Data set types supported

ES074.0

Notes:

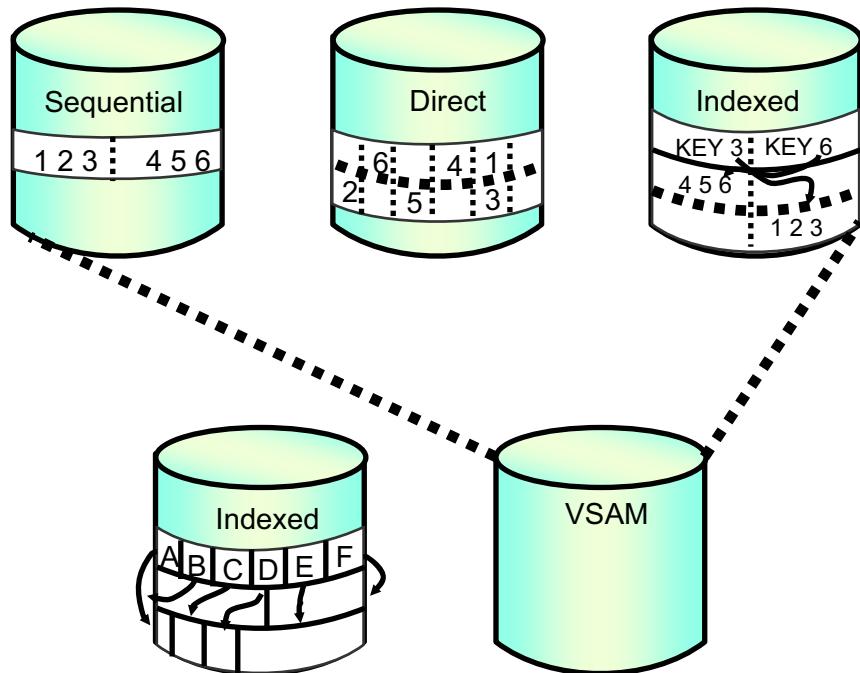
DFSMSdfp data set types

The data organization that you choose depends on your applications and the operating environment. z/OS allows you to use temporary data sets and several ways to organize files for data to be stored on permanent media, as described here.

- **VSAM data sets:** VSAM data sets are formatted differently than non-VSAM data sets. Except for linear data sets, VSAM data sets are collections of records, grouped into *control intervals*. The control interval is a fixed area of storage space in which VSAM stores records. The control intervals are grouped into contiguous areas of storage called *control areas*. To access VSAM data sets, use the VSAM access method.
- **Non-VSAM data sets:** Non-VSAM data sets are collections of fixed-length or variable-length records, grouped into blocks. To access non-VSAM data sets, use basic sequential access method (BSAM), queued sequential access method (QSAM), or basic partitioned access method (BPAM).

- **Extended-format data sets:** You can allocate both sequential and VSAM data sets in extended format on a system-managed DASD. The DASD is attached to a controller that supports Extended Platform.
- **Objects:** Objects are named streams of bytes that have no specific format or record orientation. Use the object access method (OAM) to store, access, and manage object data.
- **z/OS UNIX files:** z/OS UNIX System Services (z/OS UNIX) enables z/OS to access UNIX files. UNIX applications also can access z/OS data sets. You can use the hierarchical file system (HFS), z/OS Network File System (z/OS NFS), zSeries File System (zFS), and temporary file system (TFS) with z/OS UNIX. You can use the BSAM, QSAM, BPAM, and VSAM access methods to access data in UNIX files and directories. z/OS UNIX files are byte-oriented, similar to objects.

Data set organizations



© Copyright IBM Corporation 2011

Figure 5-13. Data set organizations

ES074.0

Notes:

Non-VSAM data sets

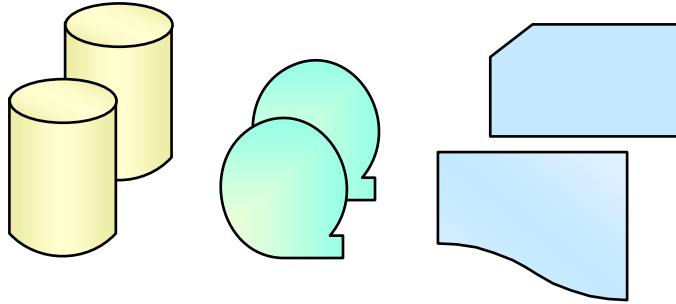
Non-VSAM data sets are collections of fixed-length or variable-length records, grouped into blocks. To access non-VSAM data sets, use BSAM, QSAM, or BPAM. There are several types of non-VSAM data sets, as explained here.

- **Sequential:** Records are written in time of arrival order. Records can be processed starting at the beginning or end of the data set. Records can be added to the end of the data set but cannot be inserted between two existing records. All device types supported.
- **Partitioned:** A group of records called a *Member* is written in sequential order and assigned a *Name* in the directory. The directory is kept in ascending sequence, but the individual members can be stored in any order. A partitioned data set must reside on

DASD and is commonly called a *Library*. All of the following data sets use partitioned organization:

- PDS
 - PDSE
 - HFS
- **Direct:** Records are written into the data set in an order determined by the program. Direct data sets can reside on DASD only.
 - **Indexed sequential:** When first loaded, records are in sequence. Various index records are built and maintained by the organization automatically. This organization is no longer recommended due to poor performance. VSAM is recommended as an alternative.
- VSAM data sets can be built via a utility, IDCAMS. If SMS is active, they can be created via JCL. VSAM data sets must reside on DASD and can be structured with four different organizations:
- **ESDS:** Resembles a sequential data set
 - **KSDS:** Resembles an indexed sequential data set
 - **RRDS:** Resembles a direct data set
 - **LDS:** Is an ESDS without any control information

Sequential organization



```
//AS DD      DSN=MY.SEQ,UNIT=SYSDA,  
//      DISP=(,CATLG),SPACE=(8900,(100,50))
```

© Copyright IBM Corporation 2011

Figure 5-14. Sequential organization

ES074.0

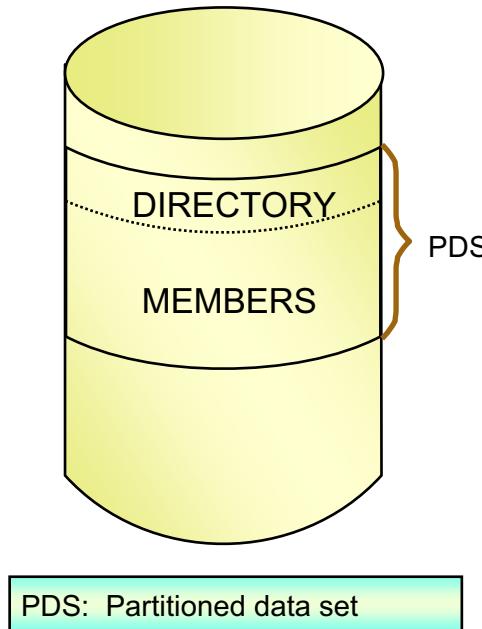
Notes:

Physical sequential data sets

Sequential data sets contain records that are stored in physical order. New records are appended to the end of the data set. You can specify a sequential data set in extended format.

- Records are written in time of arrival order.
- Records can be processed starting at the beginning or end of the data set.
- Records can be added to the end of the data sets but cannot be inserted between two existing records.
- All device types supported.
- Multiple volumes allowed.

Partitioned organization



© Copyright IBM Corporation 2011

Figure 5-15. Partitioned organization

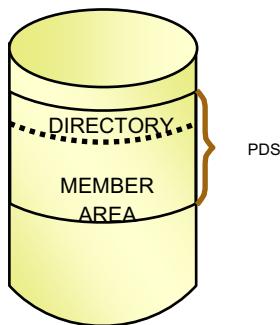
ES074.0

Notes:

Partitioned data sets contain a directory of sequentially organized members, each of which can contain a program or data. After opening the data set, you can retrieve any individual member without searching the entire data set.

- A group of records called a member is written in sequential order and assigned a *NAME* in the directory.
- A *directory* is an index that is used by the control program to locate a member in a PDS.
- The directory is kept in ascending sequence.
- Individual members can be stored in any order.
- A PDS is commonly called a *library*.
- A PDS must reside on a single volume.

PDS space specification



- **PDS**

- Directory contains member names and addresses.
- Member area contains sequential members.

```
//AP DD      DSN=MY.PDS,UNIT=SYSDA,DISP=(,CATLG),
//           VOL=SER=PAK888,SPACE=(CYL,(50,25,200))
```

© Copyright IBM Corporation 2011

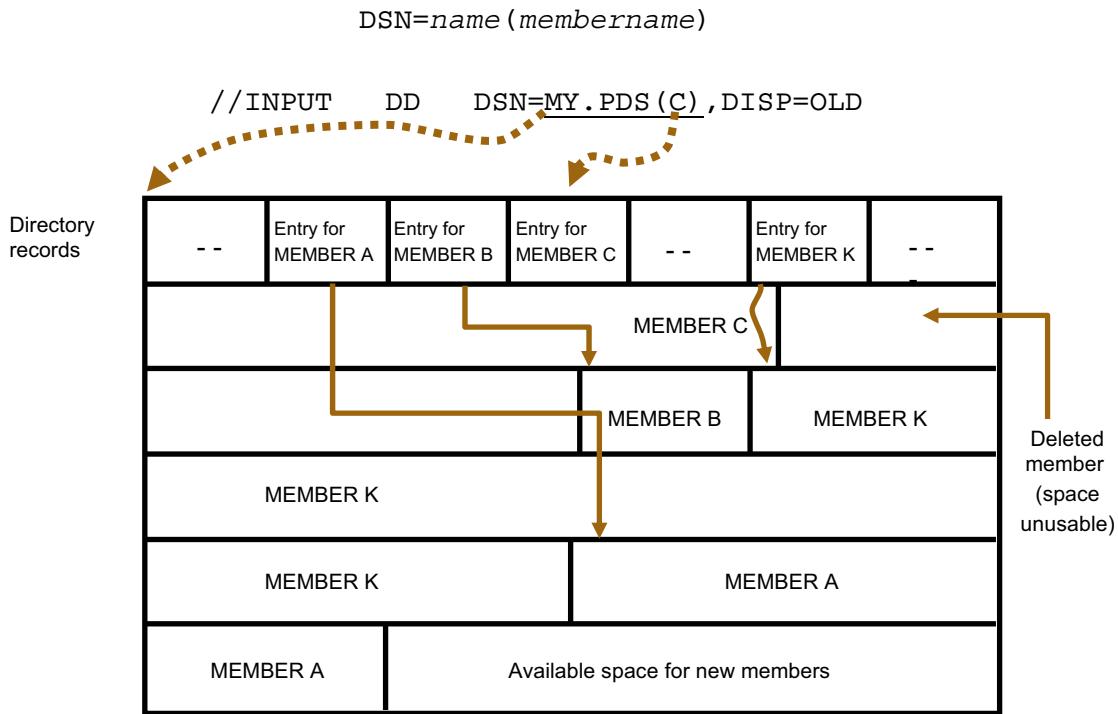
Figure 5-16. PDS space specification

ES074.0

Notes:

- The directory is made up of 256-byte blocks taken from the primary allocation.
- Secondary allocation is not used for the directory.
- It is possible to run out of space in a PDS in two ways:
 - There can be no space left in the directory.
 - There can be no space left for members.
- Note the coding of the SPACE parameter in this example. The parameter allocates 200 directory blocks. The presence of the 200 does two things:
 - The 200 makes the data set partitioned.
 - The 200 indicates how big the directory will be.

Partitioned organization: Retrieval



© Copyright IBM Corporation 2011

Figure 5-17. Partitioned organization: Retrieval

ES074.0

Notes:

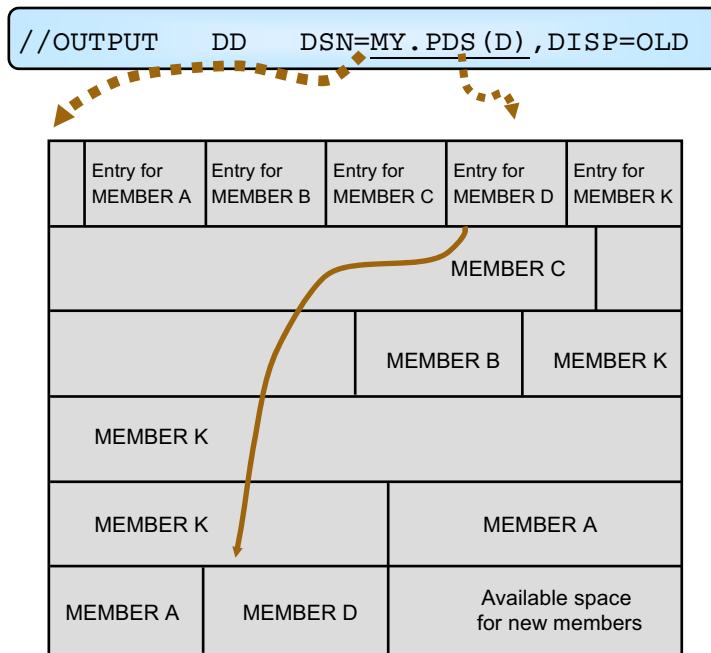
Suppose z/OS has been requested to retrieve the member C from the data set, DSN=MY.PDS.

Note the coding on the DD statement. The member C is enclosed in parentheses following the data set name on the DD statement.

z/OS will search the directory until it locates the directory entry for C. The directory entry contains the address of C.

z/OS goes to this address and retrieves C.

Partitioned organization: Addition



© Copyright IBM Corporation 2011

Figure 5-18. Partitioned organization: Addition

ES074.0

Notes:

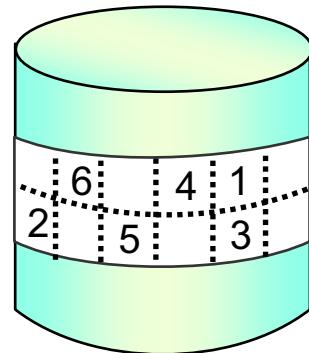
The new member named D has been added to the end of the existing PDS named MY.PDS, even though there appears to be enough room for member D following member C.

New members and modified versions of existing members are written into the available space at the end of the data set. Of course, this space can be exhausted. Members cannot be added if it is exhausted (even if sufficient space is/was introduced by deleting members).

Space formerly occupied by deleted members can be reclaimed for use by compressing the PDS. To compress a PDS, use IEBCOPY, and code the same data set name on the input and output DD statements.

Direct organization

```
//AD      DD      DSN=MY.DIR,  
//UNIT=SYSDA,DISP=(,CATLG),  
//SPACE=(CYL,50), DSORG=DA
```



Note: A secondary extent is not supported in the SPACE parameter.

© Copyright IBM Corporation 2011

Figure 5-19. Direct organization

ES074.0

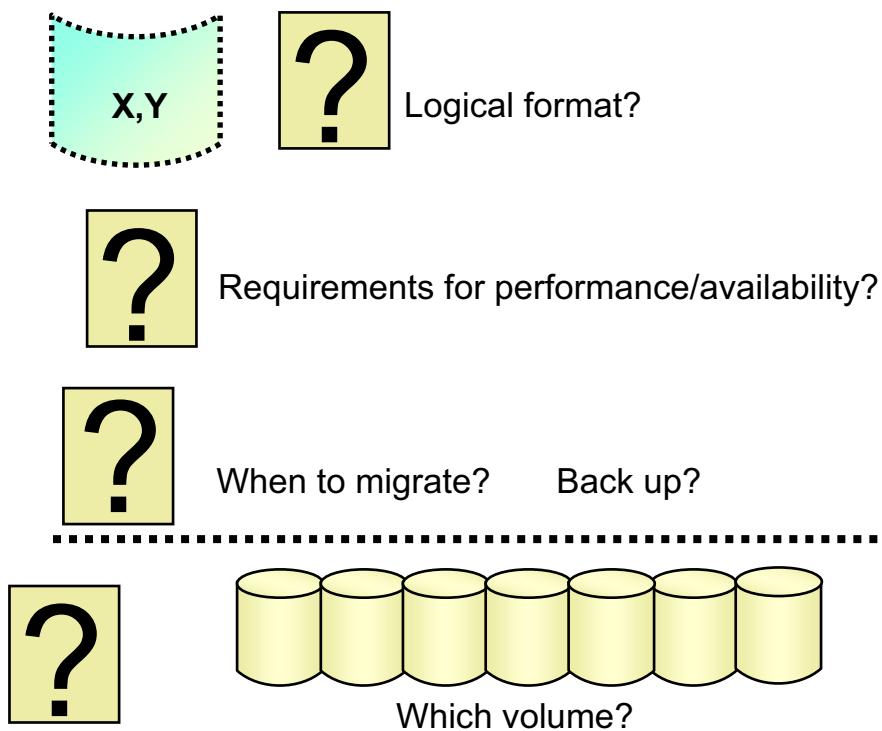
Notes:

Direct organization is designed for applications that do random processing against a data set. The processing can be read or write processing. For example, if users are randomly retrieving records from a database, the direct organization might be appropriate for that application. It is the responsibility of the application to supply the address of the record that is to be retrieved.

Two points should be noted about a data set having direct organization:

- A secondary SPACE allocation cannot be specified during creation.
- DCB=DSORG=DA must be specified when the data set is loaded.

Creating a data set: Decisions



© Copyright IBM Corporation 2011

Figure 5-20. Creating a data set: Decisions

ES074.0

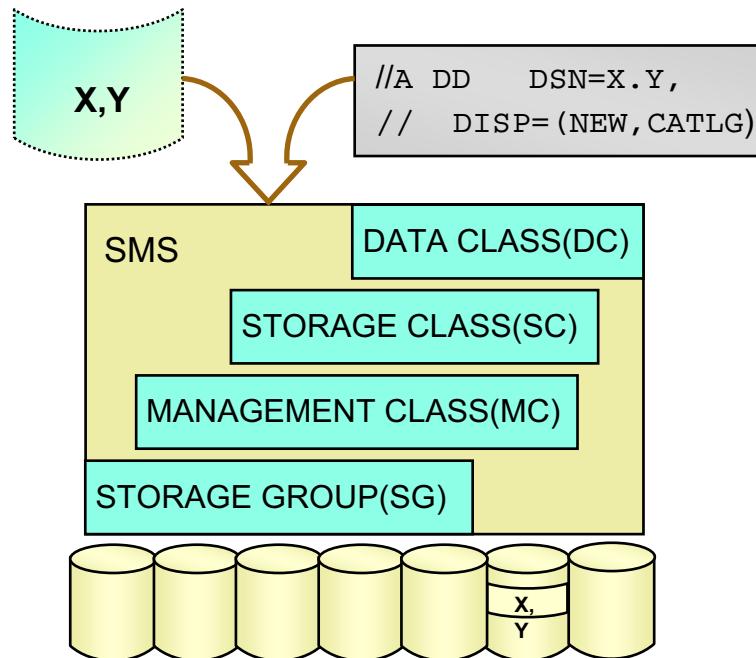
Notes:

Data processing users face numerous decisions every day in dealing with their data. In addition to dealing with issues which are unique to the data or application, they need to be aware of their installation's storage management policies. They must also deal with issues related to the format, handling, and placement of their data:

- What should the block size be? How much space will be needed?
- What type of device should I use? Should the data be cached? What about recovery?
- How often should it be backed up? Migrated? Should it be kept? Deleted?
- What volumes are available for data set placement?

If the use of data processing services is to be simplified and made available to everyone, then simpler interfaces need to be provided to the system. In particular, JCL is one area where simplifications are being made.

SMS: Automating storage management policies



© Copyright IBM Corporation 2011

Figure 5-21. SMS: Automating storage management policies

ES074.0

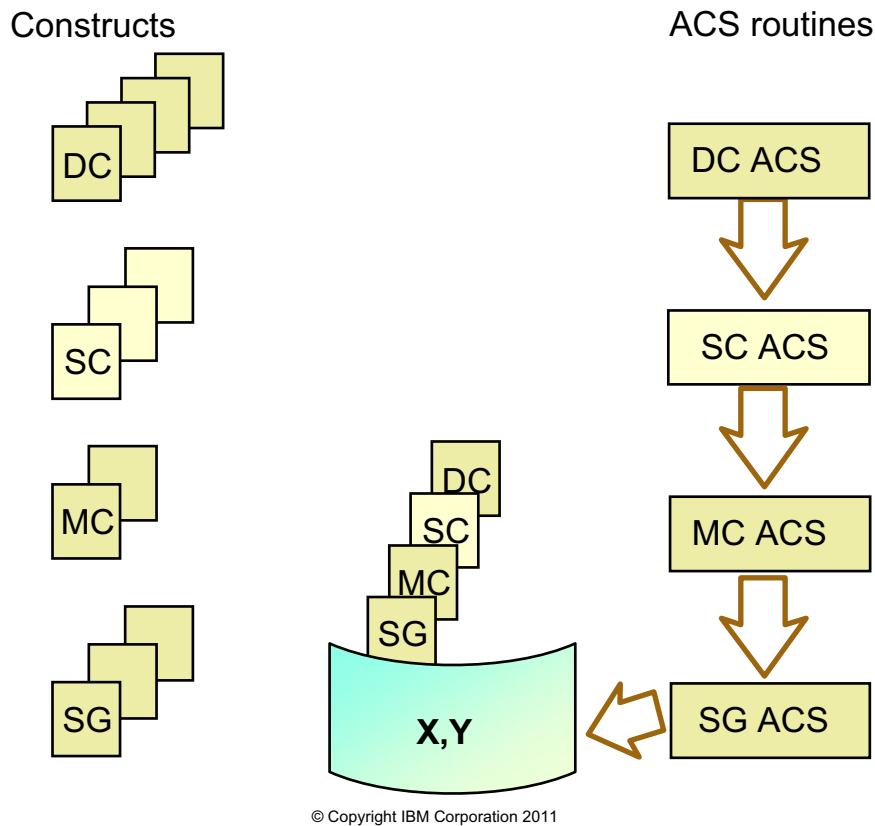
Notes:

The goal of system-managed storage is to enable any data set to be accessed by specifying nothing more than data set name and disposition. Storage Management Subsystem (SMS) design was dictated by two key objectives:

- Provide centralized control of external storage. This implies that many data management decisions can be automated through storage management policies.
 - Remove device awareness from users. Simpler JCL coding is a consequence of this.
- SMS enables an installation to implement storage management policies through four classes of predefined attributes called *constructs*.
- **DC:** The data class contains allocation attributes that describe the logical data format.
 - **SC:** The storage class contains desired performance and availability objectives. SMS uses this criteria to determine where to place a data set.
 - **MC:** The management class provides criteria that DFSMShsm uses in data set migration and backup.
 - **SG:** The storage group defines a list of volumes for data set allocation.

It is storage administrator's responsibility to set up SMS constructs; they will vary from site to site.

Constructs and ACS routines



© Copyright IBM Corporation 2011

Figure 5-22. Constructs and ACS routines

ES074.0

Notes:

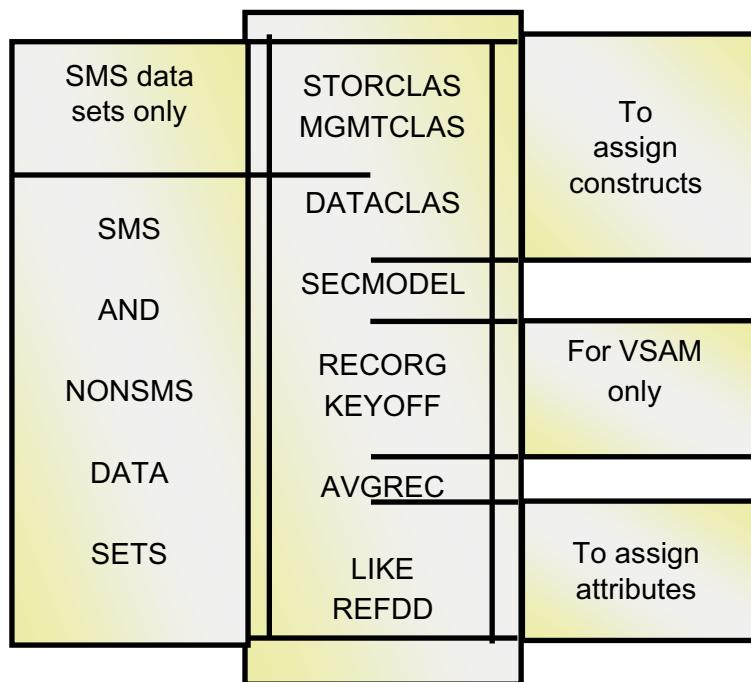
Most installations have data sets with many different formats and a multitude of performance, availability, backup, and migration requirements. Thus numerous constructs of each type need to be defined. Automatic class selection (ACS) routines are used by SMS to select and assign the proper constructs to a data set. The data set then inherits the attributes from the assigned constructs.

SMS gets control during the creation of a data set to determine if the data set is eligible to be managed by SMS. If a new data set is eligible, then the ACS routines receive control in the order shown. A maximum of one DC, one SC, and one MC can be assigned by the corresponding ACS routines.

After the ACS routines complete the construct assignments, then z/OS Scheduler Allocation gets control to complete the data set allocation.

SMS receives control not only at data set creation, but also at numerous other times during the life of a data set. SMS examines constructs in managing allocation requests, data set placement, backup and retention, and when it checks performance and availability requirements.

SMS DD parameters



© Copyright IBM Corporation 2011

Figure 5-23. SMS DD parameters

ES074.0

Notes:

SMS provides nine DD statement parameters that can be used to define new data sets. If SMS is active, these parameters can simplify JCL DD statement coding. That is, several old JCL DD parameters can be replaced by fewer parameters from the list in the visual; however, if SMS is inactive, all of these parameters will be ignored.

`DATACLAS`, `STORCLAS`, and `MGMTCLAS` are DD statement parameters which are used to assign a DC, SC, and MC respectively. (Therefore, the DC, SC, and MC can be assigned via ACS routines as well as JCL!)

`STORCLAS` and `MGMTCLAS`, apply only to SMS-managed data sets, whereas `DATACLAS` and the remaining parameters apply to both SMS-managed and non-SMS-managed data sets. `RECORG` and `KEYOFF` apply only to VSAM data sets.

The `LIKE` and `REFDD` parameters can be coded when creating a new data set to specify the allocation attributes to assign. `LIKE` tells the system to use the allocation attributes from an existing catalogued data set. `REFDD` tells the system to use the allocation attributes from a data set defined earlier in the job.

`SECMODEL` specifies a model RACF profile which RACF uses to create a discrete profile to protect the data set. Consult the *z/OS MVS JCL Reference* for further details.

Space allocation: AVGREC

```
//N DD DSN=X.Y,RECFM=FB,
// LRECL=80,DISP=(,CATLG),
// SPACE=(890,(1000,1000)),AVGREC=U
```



Allocate space for 1000 80-byte logical records.

© Copyright IBM Corporation 2011

Figure 5-24. Space allocation: AVGREC

ES074.0

Notes:

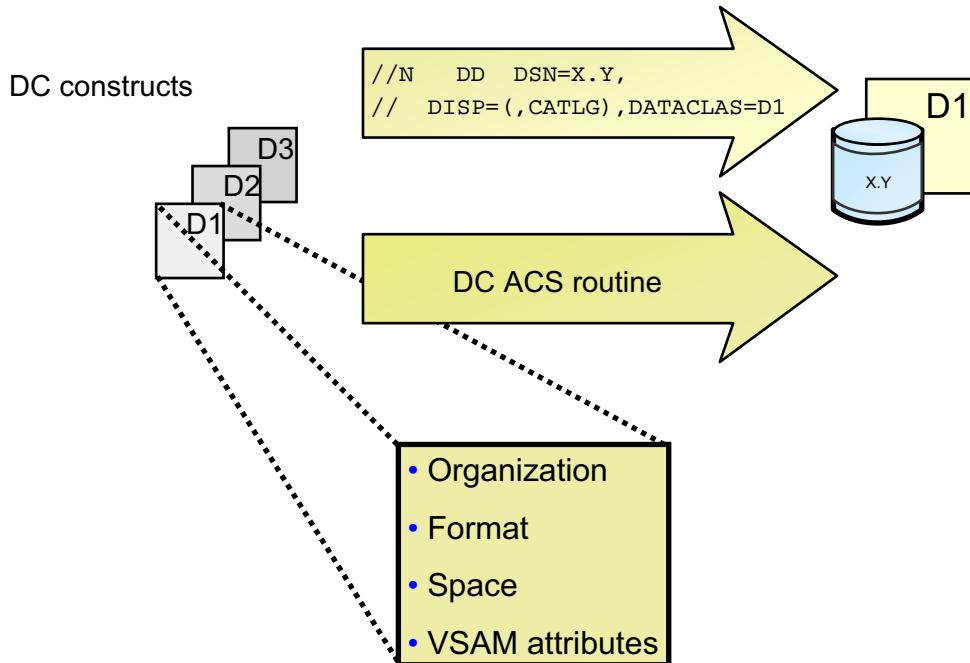
If AVGREC is coded with SPACE on the DD statement, SPACE specifies space in multiples of average record lengths instead of block lengths. Average record length is the average number of bytes per record. Three AVGREC values determine how the SPACE parameter's unit of allocation is interpreted:

- **AVGREC = U:** The unit of allocation is average record length in bytes.
- **AVGREC = K:** The unit of allocation is average record length in Kbytes.
- **AVGREC = M:** The unit of allocation is average record length in Mbytes.

The average record length in the SPACE parameter should always be the same as the LRECL for fixed-length records. In all cases, the allocated space will be rounded up to the next whole track.

In the example in the visual, AVGREC=U is coded, therefore the system allocates primary space for 1000 80-byte records. If necessary, the system provides secondary extents containing room for 1000 80-byte logical records. Since RECFM specifies fixed-length records, the SPACE parameter's unit of allocation is the same as LRECL.

Assigning a data class



© Copyright IBM Corporation 2011

Figure 5-25. Assigning a data class

ES074.0

Notes:

The data class (DC) contains values for attributes such as LRECL, RECFM, RECORG, KEYLEN, KEYOFF, SPACE, and EXPDT. BLKSIZE is not specified through the DC. If BLKSIZE is omitted, the system will use the system-determined block size.

DC can be assigned to both SMS- and non-SMS-managed data sets. The DC can be assigned by the DC ACS routine, the DATACLAS DD parameter, and by other methods (for example, ISPF/PDF, TSO/E, and IDCAMS). If it is not assigned by any other method, the ACS routine will assign it. The data class D1 can be assigned via the DATACLAS parameter:

```
//N DD DSN=X.Y,DISP=(,CATLG),DATACLAS=D1
```

A DC attribute can be overridden by coding that attribute on the DD statement. For example if D1 contains RECFM=FB, the user can request RECFM=F by coding

```
//N DD DSN=X.Y,DISP=(,CATLG),DATACLAS=D1,RECFM=F
```

X.Y will be assigned RECFM=F but will inherit all *remaining* D1 attribute values.

Data class list

```

Panel List Utilities Scroll Help
-----
                               DATA CLASS LIST
Command ===>                                         Scroll ==> HALF
                                                       Entries 1-4 of 4
                                                       Data Columns 3-9 of 47
CDS Name : DFSMS.SCDS.ALT

Enter Line Operators below:

LINE      DATACLAS
OPERATOR  NAME     RECORC  RECFM   LRECL   KEYLEN  KEYOFF  AVGREC  AVG
--- (1) ---  -- (2) -- - (3) -- - (4) - - (5) - - (6) -- - (7) -- - (8) -- - (9) -
          DB2STRIP  --
          DCSOGEI   --
          LOGGER    LS
          TESTDC   --
-----           BOTTOM OF DATA -----

```

© Copyright IBM Corporation 2011

Figure 5-26. Data class list

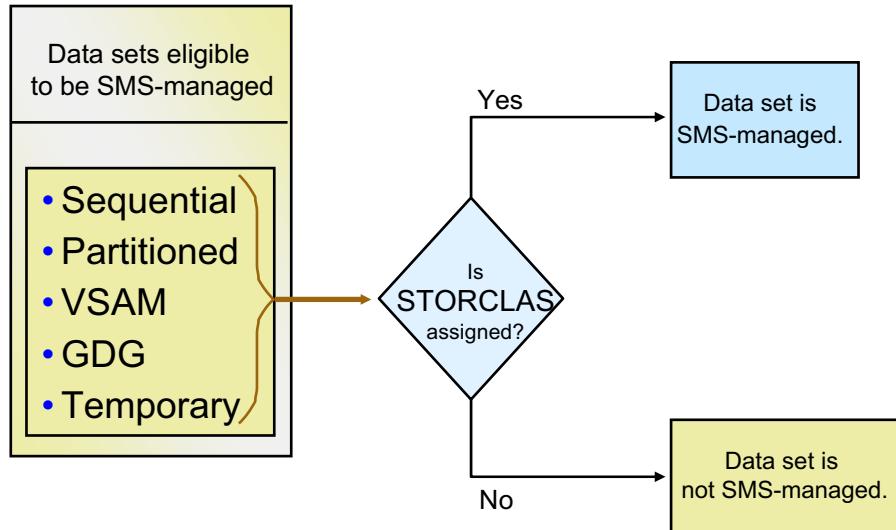
ES074.0

Notes:

Since the DC can be assigned via the DATACLAS keyword, z/OS users need to have a method of listing the data classes along with their attribute values. The Interactive Storage Management Facility (ISMF) provides this capability.

ISMF generated the screen in the visual. The screen displays the data classes and summarizes their attribute values. Thus, z/OS users can quickly scan the list to see what data classes are available.

SMS-managed data sets



© Copyright IBM Corporation 2011

Figure 5-27. SMS-managed data sets

ES074.0

Notes:

In order for a data set to be SMS-managed, the data set must satisfy two criteria:

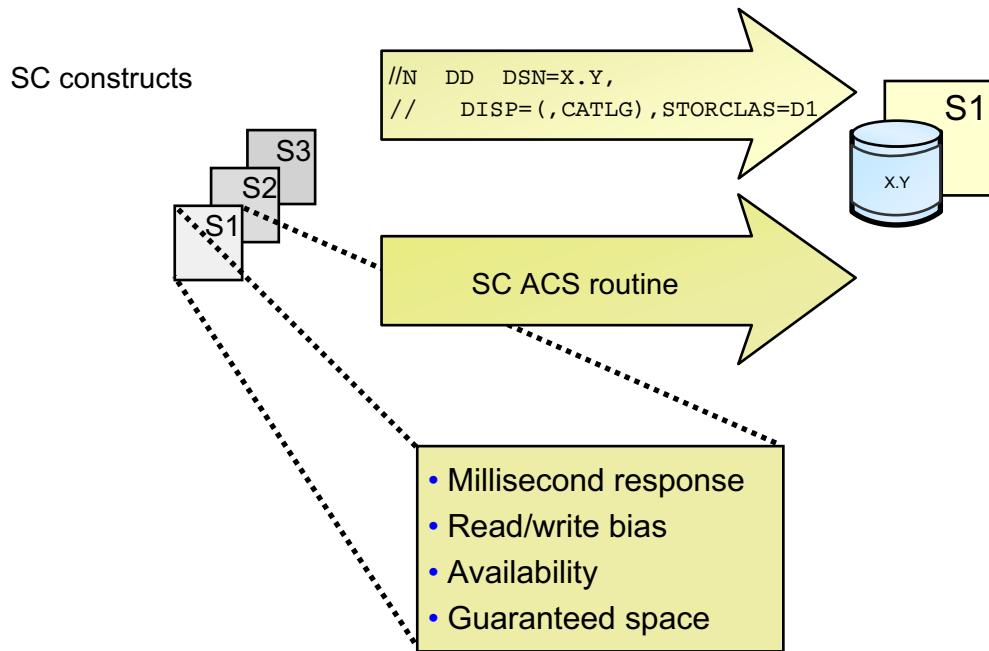
- The data set must be eligible to be SMS-managed.
- The data set must have a storage class (SC) assigned.

The categories of data sets above are eligible to be managed by SMS, but not all data sets are eligible. The following categories are not eligible:

- Data sets not cataloged in an ICF catalog
- Indexed sequential access method (ISAM) data sets
- SYSOUT data sets
- Instream data sets
- Unmovable data sets
- Data sets allocated with absolute track allocation
- Data sets on a mass storage (MSS) volume
- Generation data group (GDG) model DSCBs
- VSAM spaces
- VSAM catalogs or CVOLS (control volumes - old catalog type in MVS)

Note that tape data sets are sequential. Thus they are eligible to be SMS-managed.

Assigning a storage class



© Copyright IBM Corporation 2011

Figure 5-28. Assigning a storage class

ES074.0

Notes:

The storage class (SC) provides criteria that SMS uses in determining where to place a data set. Desired response time, the read/write bias (whether reads or writes are predominant), the importance of the data set (with respect to availability), and attributes relating to optical volumes and multivolume data sets are specified through the SC.

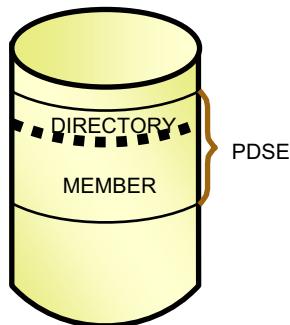
The storage class can be assigned by the SC ACS routine, the STORCLAS DD parameter, and by other methods (for example, ISPF/PDF, TSO/E, and IDCAMS). The assignment of the storage class S1 can be requested through the STORCLAS parameter:

```
//N DD DSN=X.Y,DISP=(,CATLG),STORCLAS=S1
```

However, the storage administrator can elect to honor this request or allow the SC ACS routine to override the request and make the assignment.

A management class can be requested by coding MGMTCLAS=M1. The only way to assign a storage group is to allow the SG ACS routine to assign it. SMS then selects the volume from this group.

Partitioned organization: PDSE



- PDSE
 - Must be SMS-managed
 - Has directory and member area
 - More efficient than a PDS

```
//BP DD DSN=MY.PDSE,DISP=(,CATLG),
// DATACLAS=SPDS,SPACE=(CYL,(100,20,500)),
// DSNTYPE=LIBRARY
```

© Copyright IBM Corporation 2011

Figure 5-29. Partitioned organization: PDSE

ES074.0

Notes:

A PDSE is an SMS-managed data set that has a partitioned organization. That is, it has a directory and member area containing members.

A PDSE is created in the same way as a PDS except that the parameter DSNTYPE=LIBRARY must be added to the DD statement to create a PDSE. Otherwise, a PDS will be created. See the example in the visual.

However, a PDSE has several characteristics that make it superior to a PDS:

- It provides better performance (efficiency) than a PDS.
- It never needs to be compressed.
- You do not encounter directory shortages with a PDSE.
- There is less chance of making it unusable during an update.
- It allows multiple members to be added at the same time.

Differences: PDS versus PDSE

	PDS	PDSE
SMS-managed?	Not required	Must be
Retrieval from?	Usually DASD	Often storage
Can hold programs?	Yes	Yes
Need to compress?	Yes	No
Directory shortages?	Possible	No
Access protection	Data set level	Member level

© Copyright IBM Corporation 2011

Figure 5-30. Differences: PDS versus PDSE

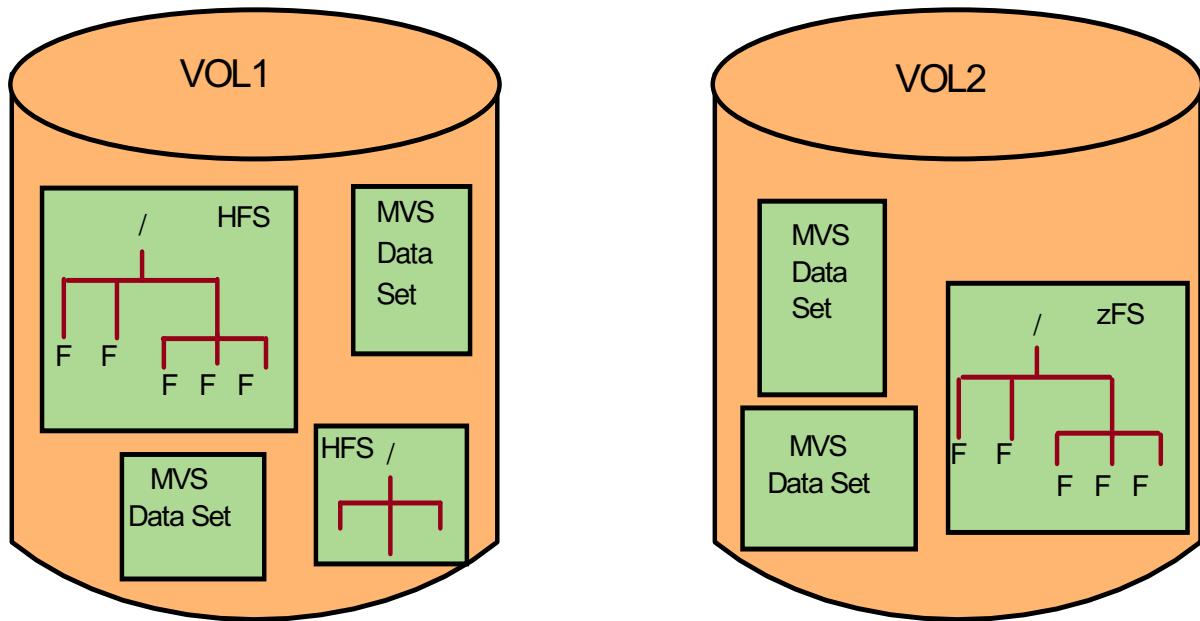
ES074.0

Notes:

PDSs differ from PDSEs in the following ways:

- A PDSE must be SMS-managed. You could also have an SMS-managed PDS, but that is not required.
- Executable programs can be stored in PDSs or PDSEs; however, active executable programs in PDSEs are often fetched from virtual storage instead of DASD. Executable programs in PDSs are often fetched directly from DASD.
- Executable programs can be stored in both PDSs and PDSEs.
- PDSEs never need to be compressed, whereas PDSs do.
- You can run out of space in a PDS directory, but not a PDSE directory.
- If a user is accessing a PDS with `DISP=OLD`, anyone else who wants to access it at the same time will be placed in a wait. Multiple users can access a PDSE at the same time if each user codes `DISP=SHR`. In particular, users can add multiple members at the same time.

HFS and zFS data sets



© Copyright IBM Corporation 2011

Figure 5-31. HFS and zFS data sets

ES074.0

Notes:

A hierarchical file system may be in a data set type called HFS (older) or may be in a zFS (newer) data set.

HFS data set can span many DASD volumes. If spanned, they must be SMS-managed. However, while HFS always has a single file system per data set, zFS may have multiple file systems in a single data set. These data sets are called *aggregates* and are a collection of data sets.

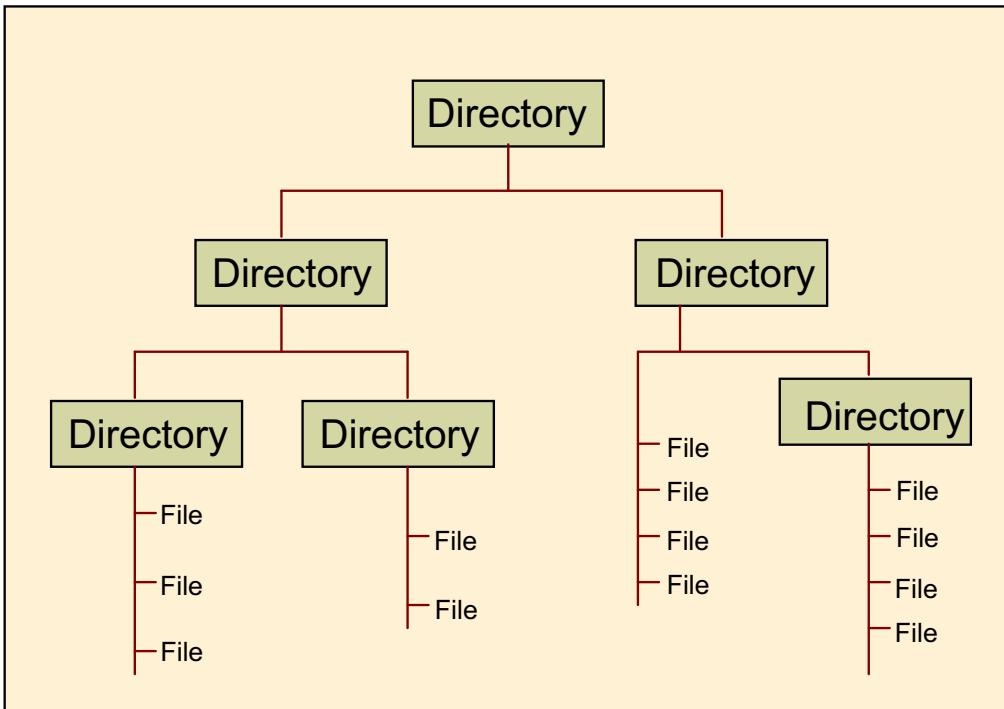
HFS data sets are PDSE-like data sets. They are allocated like any other MVS data set. Although an HFS data set is an MVS data set, an MVS OPEN request will fail. Users must use UNIX System Services to access data in such a data set.

zFS is a file system that is different from HFS. The administration view of zFS is different from HFS, and zFS file systems are created in a different manner than HFS file systems are created. In general, however, the application view of zFS is the same as the application view of HFS.

The same APIs and commands are used for zFS as are used for HFS. Once the zFS file system is mounted, it is almost indistinguishable from a mounted HFS file system.

File system structure

zFS/ HFS data set



© Copyright IBM Corporation 2011

Figure 5-32. File system structure

ES074.0

Notes:

The z/OS UNIX file system is hierarchical in structure in the same way as a UNIX file system. All files are members of a directory, and each directory is in turn a member of another directory at a higher level in the hierarchy. The highest level in the hierarchy is called the *root directory*.

A file in the hierarchical file system is called an *HFS file*. HFS files are byte-oriented, and there is no concept of a record structure.

A file system is contained in an MVS data set type called a *HFS data set* or *zFS*.

The hierarchical file system in z/OS UNIX is similar to the file system in Linux and Windows.

As with other UNIX file systems, a path name identifies a file and consists of directory names and a file name. A fully qualified file name, which consists of the name of each directory in the path to a file plus the file name itself, can be up to 1023 bytes long.

The path name is constructed of individual directory names and a file name separated by the forward-slash character, for example:

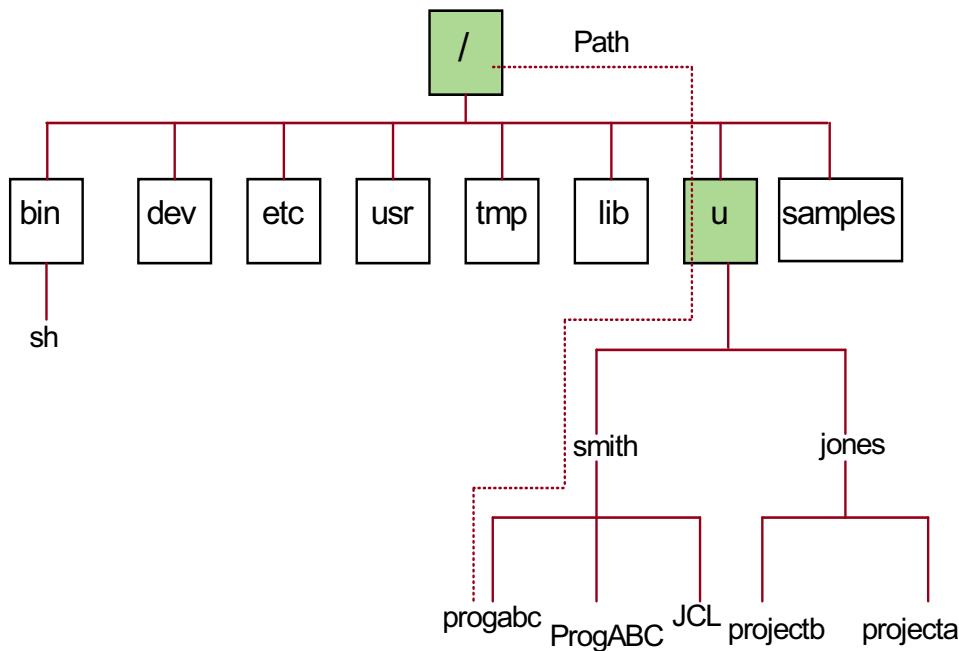
/dir1/dir2/dir3/MyFile

Like UNIX, z/OS UNIX is case-sensitive for file and directory names.

For example, in the same directory, the file *MYFILE* is a different file than *MyFile*.

The files in the hierarchical file system are sequential files and are accessed as bytestreams. A record concept does not exist with these files other than the structure defined by an application.

Path name and file name



© Copyright IBM Corporation 2011

Figure 5-33. Path name and file name

ES074.0

Notes:

The set of names required to specify a particular file in a hierarchy of directories is called the *path* to the file. The path is specified as a path name.

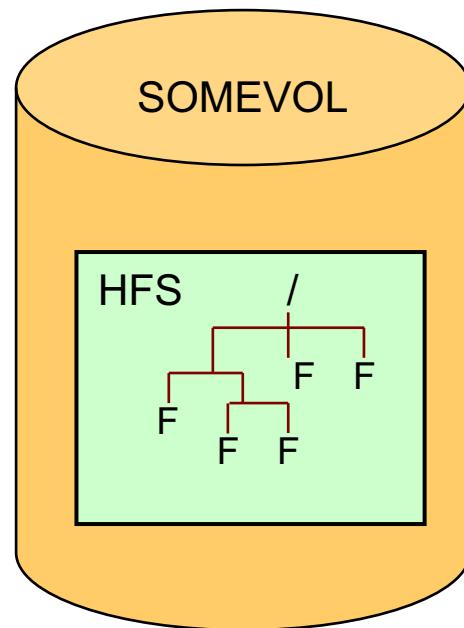
A path name can be absolute or relative. An absolute path name is a sequence which begins with a slash (/) for the root, followed by one or more directories, and ends with the file name. A relative path name is a path name which is relative to the working directory. A relative path name does not start with a slash.

A path name can be up to 1023 characters long, including all directory names, separating slashes, and the file name.

A file name can be up to 255 characters long and can consist of uppercase and lowercase letters. z/OS UNIX is case sensitive, which means a file called *ProgABC* is different from *progabc*. File names can include extensions which identify the contents of a file, for example, *proga.c* and *name.lst*.

HFS data set characteristics

- DSNTYPE=HFS
 - Max 123 extents
- Spanned data sets supported
 - If spanned - must be SMS
- Non-spanned data set supported
 - On non-SMS managed volumes
- Maximum file size 8 TB



© Copyright IBM Corporation 2011

Figure 5-34. HFS data set characteristics

ES074.0

Notes:

The hierarchical file system (HFS) consists of one root file system and multiple user file systems. The root file system is the base for HFS and is at the top of the hierarchy.

- HFS data sets are allocated as z/OS data sets DSNTYPE=HFS.
- HFS data sets can only be opened by UNIX System Services kernel operations.
- HFS data sets can reside on the same volumes as other z/OS data sets.
- Non-SMS-managed HFS data sets are often used.
- The size of the data set is limited to 2 GB if single-volume.
- A multivolume HFS can span up to a maximum of 59 volumes.
- HFS can have up to a maximum of 123 extents per volume.
- A multivolume HFS can have up to a maximum of 255 total extents for all volumes.

zFS file systems

- An aggregate is a VSAM linear data set (LDS).
- An aggregate contains one or more zFS file systems.
- There are two types of aggregates.
 - **HFS compatibility mode:** Contains one zFS file system
 - **Multiple file system mode:** Contains one or more zFS file systems
- Space sharing between file systems in same aggregate.

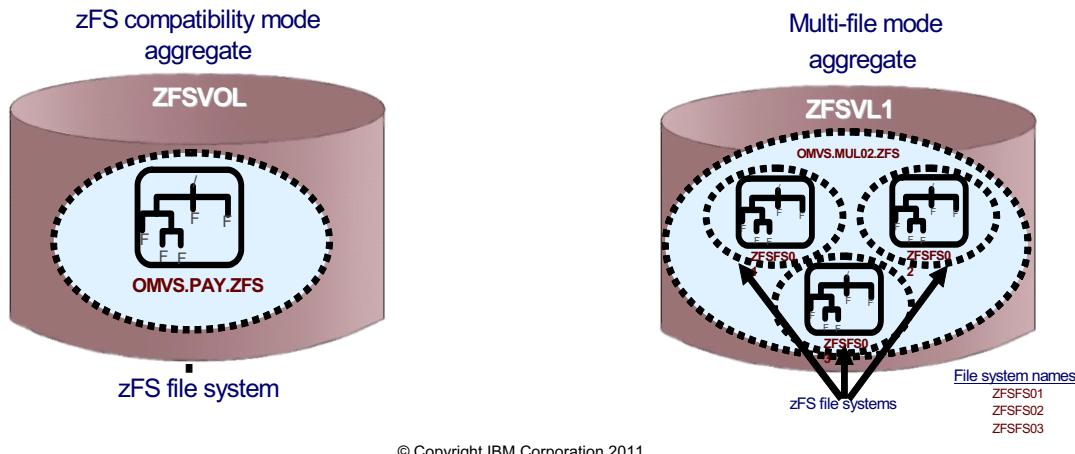


Figure 5-35. zFS file systems

ES074.0

Notes:

A *zFS aggregate* is a data set that contains zFS file systems. The aggregate is a VSAM linear data set (VSAM LDS) and is a container that can contain one or more zFS file systems.

An aggregate can only have one VSAM LDS, but it can contain an unlimited number of file systems. The name of the aggregate is the same as the VSAM LDS name.

A zFS aggregate that contains only a single zFS file system is called a *compatibility mode aggregate*.

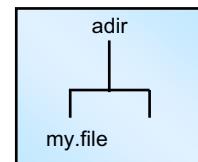
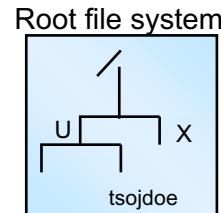
Compatibility mode aggregates are more like HFS.

Aggregates that contain multiple file systems are called *multi-file system aggregates*.

HFS data set

- HFS creation

```
//FS DD DSN=TSOJDOE.OMVS.HFS,
//           DISP=(,CATLG),DATACLAS=SPDS,
//           SPACE=(CYL,(80,40,1)),
//           DSNTYPE=HFS
```



TSOJDOE.OMVS.HFS

- z/OS UNIX file creation

```
//F DD PATH='/u/tsojdoe/adir/my.file',
//PATHMODE=(SIRWXU,SIWGRP),PATHDISP=(KEEP,DELETE),
//PATHOPTS=(OWRONLY,O_CREAT,O_EXCL)
```

© Copyright IBM Corporation 2011

Figure 5-36. HFS data set

ES074.0

Notes:

Hierarchical file system data sets:

- Have partitioned organization
- Have the same structure as a PDSE
- Might or might not be SMS-managed
- Contain mountable file systems used by z/OS UNIX Services

z/OS UNIX Services supports UNIX applications.

To create an HFS data set, specify the usual DD statement parameters which are necessary to create any z/OS data set. In addition, specify DSNTYPE=HFS to ask z/OS to create an HFS data set.

The additional PATH_____ parameters shown in the visual must be specified to create an HFS file in the HFS data set.

- PATH specifies the name of the HFS file to be created.
- PATHMODE specifies the file access attributes.
- PATHDISP specifies the file disposition at end of step.
- PATHOPTS specifies the file access and status.

JCL support

DSNTYPE=HFS | PIPE

PATH=*pathname*

PATHDISP=(*normal-termination-disp*,*abnormal-term-disp*)

PATHMODE=(*file-access-attribute*,*file-acc-attr*, . . .)

PATHOPTS=(*file-option*,*file-option*, . . .)

© Copyright IBM Corporation 2011

Figure 5-37. JCL support

ES074.0

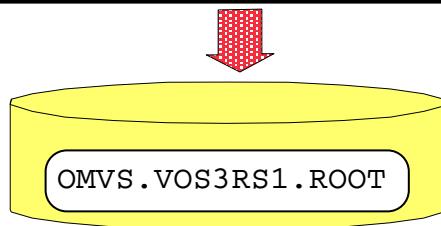
Notes:

One way of using a file in the hierarchical file system is to refer to it by JCL. The following changes have been made to support hierarchical files:

- **DSNTYPE=HFS/PIPE:** Specify HFS to create a hierarchical file system data set, or PIPE to specify a named pipe first in, first out (FIFO).
- **PATH:** Identifies a file in the hierarchical file system. The JCL support allows a pathname to be up to 254 characters long, not including the slashes. If the pathname contains lowercase letters, the name must be enclosed by single quotes.
- **PATHDISP:** Specifies the disposition of a hierarchical file after normal or abnormal end. It can be KEEP or DELETE.
- **PATHMODE:** Specifies the file access attributes when the file specified in the PATH parameter is created. The file access attributes define who has access to read, write, or execute the file.
- **PATHOPTS:** Specifies the access and status for a file. There are two file options groups: the access group and the status group.

Allocate HFS: JCL example

```
//OMVSHFSA    JOB   (....)
//STEP1        EXEC   PGM=IEFBR14
//MKROOT       DD     DSNAME=OMVS.VOS3RS1..ROOT,
//                           SPACE=(CYL,(600,100,1)),
//                           UNIT=3390,
//                           DSNTYPE=HFS,
//                           DCB=DSORG=PO,
//                           DISP=(NEW,CATLG,DELETE),
//                           STORCLASS=STANDARD
```



© Copyright IBM Corporation 2011

Figure 5-38. Allocate HFS: JCL example

ES074.0

Notes:

This shows an example of how to allocate an HFS data set for the root file system.

The important JCL keyword here is DSNTYPE=HFS.

Note that an HFS can be either SMS- or non SMS-managed but must be cataloged for the mount to succeed.

Allocate and format a zFS aggregate

```
//USERIDA JOB , 'Compatibility Mode', Example:
//           CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//AMSDUMP DD SYSOUT=H
//DASD0 DD DISP=OLD,UNIT=3390,VOL=SER=PRV000
//SYSIN DD *
      DEFINE CLUSTER (NAME(OMVS.PRV.COMPAT.AGGR001) -
                      VOLUMES(PRV000) -
                      LINEAR CYL(25 0) SHAREOPTIONS(2)) -
/*
//CREATE EXEC PGM=IOEAGFMT,REGION=0M,
// PARM=' -aggregate OMVS.PRV.COMPAT.AGGR001 -compat'
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
/*
```



© Copyright IBM Corporation 2011

Figure 5-39. Allocate and format a zFS aggregate

ES074.0

Notes:

The **IOEAGFMT** utility is used to format an existing VSAM LDS as a zFS aggregate. All zFS aggregates must be formatted before use (including HFS compatibility mode aggregates).

IOEAGFMT does not require the zFS physical file system to be active on the system. The size of the aggregate is as many 8 KB blocks as will fit in the primary allocation of the VSAM LDS.

To extend it to its secondary allocation (assuming it has a secondary allocation), use the **zfsadm grow** command.

JCL example with a UNIX file

```

//LINK      JOB (XX,YY,ZZ),MSGCLASS=H,CLASS=A,
//                  MSGLEVEL=(1,1)
//LKED      EXEC PGM=IEWBLINK,
//                  PARM='LIST,REUS,RENT,LET,CASE=MIXED'
//SYSPRINT DD *
//INLIB      DD DSN=POSIX.LOADLIB,DISP=SHR
//SYSLMOD   DD PATH='/u posix/llib',
//                  PATHOPTS=(OWRONLY,OCREATE,OTRUNC),
//                  PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//SYSLIN    DD *
INCLUDE INLIB(PAYRLL)
ENTRY CEESTART
NAME payrll(R)
/*

```

© Copyright IBM Corporation 2011

Figure 5-40. JCL example with a UNIX file

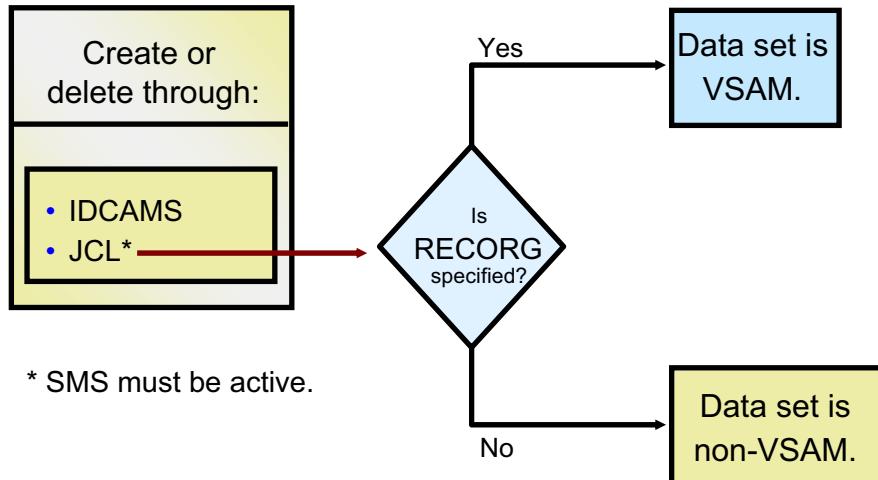
ES074.0

Notes:

This JCL example shows a program in an MVS data set being relinked to reside in the hierarchical file system.

- PATH defines the path name of the directory where the program `payrll` will reside when link-edited. The path name is in lowercase letters and therefore enclosed by single quotes.
- PATHOPTS
 - OWRONLY specifies the file will be opened for write.
 - OCREATE specifies that if the file does not exist, it will be created. If it exists, it will be reused.
 - OTRUNC specifies that the file length will be truncated to zero if the file already exists.
- PATHMODE
 - SIRWXU specifies that the file owner has read, write, and execute permission.
 - SIRWXG specifies that file group has read, write, and execute permission.
 - SIRWXO specifies that all other users have read, write, and execute permission.

VSAM data sets



© Copyright IBM Corporation 2011

Figure 5-41. VSAM data sets

ES074.0

Notes:

It is possible to create VSAM data sets having four organizations:

- **KSDS:** Key-sequenced data set. Contains records in ascending collating sequence and can be accessed by a field or called by a key, or by a relative byte address.
- **ESDS:** Entry-sequenced data set. Contains records in the order in which they were entered. Records are added to the end of the data set and can be accessed sequentially or randomly.
- **RRDS:** Relative record data set. Contains records in order by relative record number and can be accessed only by this number.
- **LDS:** Linear data set. Contains data that has no record boundaries. Linear data sets contain no control information.

VSAM data sets can be created or deleted using either IDCAMS or JCL. SMS must be active to use JCL. All VSAM data sets must be cataloged.

To create a VSAM data set via JCL, specify the RECORG parameter (either in a data class or on a DD statement). RECORG can have four values:

- RECORG=KS says to create a VSAM KSDS
- RECORG=ES says to create a VSAM ESDS
- RECORG=RR says to create a VSAM RRDS
- RECORG=LS says to create a VSAM LDS

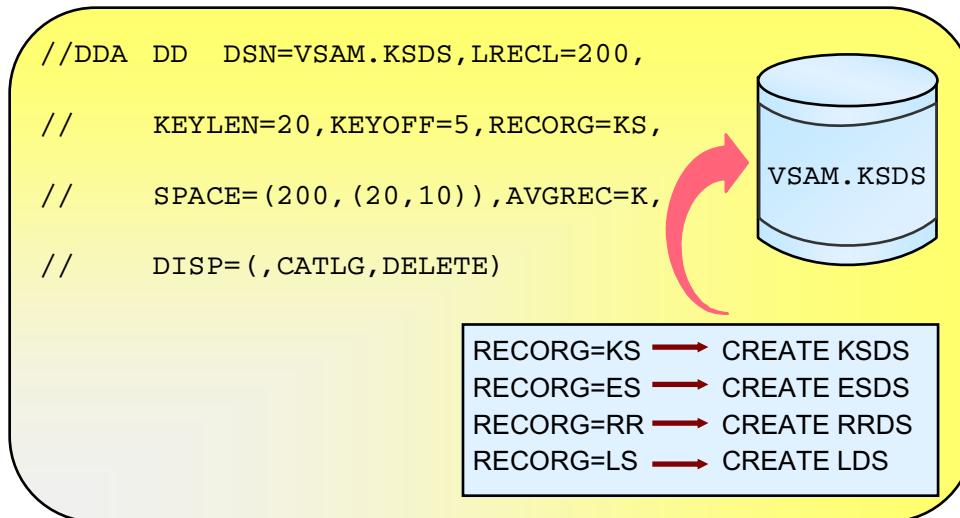
The JCL DSN parameter defines the VSAM cluster name. The system generates the data component name from the cluster name. If a KSDS is being defined, the system also generates an index component name from the cluster name.

If RECORG is unspecified or SMS is inactive, the system creates a non-VSAM data set.

A VSAM data set can be deleted by coding `DISP=(____,DELETE)` when SMS is active. Therefore, temporary VSAM data sets can be defined and deleted using JCL. They can be SMS-managed or non-SMS-managed, but they must be catalogued.

If SMS is inactive, `DISP=(____,DELETE)` is ignored. Thus, if SMS is inactive, VSAM data sets can neither be created nor deleted through JCL.

VSAM: RECOR and KEYOFF



© Copyright IBM Corporation 2011

Figure 5-42. VSAM: RECOR and KEYOFF

ES074.0

Notes:

If SMS is active, the JCL shown in the visual will create and catalog a VSAM KSDS named VSAM.KSDS.

- RECOR=KS and DISP=(NEW, ____) tell the system to create a VSAM KSDS.
- KEYLEN=20 and KEYOFF=5 tell the system that each record contains a key of length 20 bytes, which is offset 5 bytes into the record.
- AVGREC=K and SPACE=(200, (20,10)) tell the system that the primary extent should contain space for $20 * 1024 = 20,480$ records having an average length of 200 bytes.
Note that LRECL = 200. (If this data set contains fixed length records, LRECL must equal the unit of allocation for the SPACE parameter!)

If SMS is inactive, KEYOFF, RECOR, and AVGREC are ignored, and the system creates a non-VSAM data set. Therefore, always check to be certain that a VSAM data set was created.

Changes to job processing

- **VOL=SER:** Ignored by SMS
- **DISP=(,KEEP) :** Means DISP=(,CATLG) to SMS
- **SMS-managed data sets:** Cataloged at creation
- **JOBCAT and STEPCAT:** No longer used *
- **Esoteric UNITNAMES:** No longer used
- **DASD USE attributes:** No longer used

- * **Note:** JOBCAT/STEPCAT do not work as of z/OS V1R7. They are kept for compatibility reasons but not honored.

© Copyright IBM Corporation 2011

Figure 5-43. Changes to job processing

ES074.0

Notes:

There are several important changes to job processing in the SMS environment:

- If VOL=SER is coded when an SMS-managed data set is created, SMS ignores VOL=SER. SMS will select a volume from the storage group construct.
- SMS-managed, permanent data sets must be cataloged in an ICF catalog. Thus, if DISP=(,KEEP) is coded to create and keep a data set, SMS will catalog it.
- SMS-managed, permanent data sets are cataloged at start but not end of step.
- If a job containing a JOBCAT or STEPCAT DD statement references an SMS-managed data set, SMS generates a JCL error. Thus, that step and all subsequent steps will fail.
- Esoteric unit names like SYSDA are not used for SMS data sets. The storage groups assigned by the SC ACS routine contain eligible volumes for SMS-managed data sets. Esoteric unit names are still used for non-SMS-managed data sets.
- The DASD use attributes PUBLIC, PRIVATE, and STORAGE have no meaning for SMS-managed data sets. The storage group supplies volumes for allocation. The three use attributes are still used for non-SMS-managed data sets.

Checkpoint (1 of 4)

1. What is the maximum block size supported in z/OS?
 - a. 4096
 - b. 32756
 - c. 32760

2. With DCB=(BLKSIZE=3200,LRECL=80,RECFM=FB), how many records do you have in each block?

3. With DCB=(BLKSIZE=3200,LRECL=80,RECFM=F), how many records do you have in each block?

4. Which of the following are valid record formats?
 - a. RECFM=F
 - b. RECFM=FB
 - c. RECFM=FV

© Copyright IBM Corporation 2011

Figure 5-44. Checkpoint (1of 4)

ES074.0

Notes:

Checkpoint (2 of 4)

5. How do you let the system use the optimum block size?
6. What is the SMS JCL parameter to assign standard data set allocation attributes?
7. Name different data set types.
8. Name four VSAM organizations.
9. True or False: In order for a data set to be SMS managed, the data set must have *all* of the following: data class, storage class, and management class.

© Copyright IBM Corporation 2011

Figure 5-45. Checkpoint (2 of 4)

ES074.0

Notes:

Checkpoint (3 of 4)

10. What are the AVGREC values to determine how the SPACE parameter's unit of allocation is interpreted?
11. Match DC, SC, MC, and SG to the following:
 - a. Defines backup and retention requirements
 - b. Defines model allocation characteristics for data sets
 - c. Creates logical groupings of volumes to be managed as a unit
 - d. Defines performance and availability goals
12. A UNIX file name can be up to how many characters long?
13. True or False: A PDSE must be SMS-managed.
14. True or False: An HFS must be SMS-managed.

© Copyright IBM Corporation 2011

Figure 5-46. Checkpoint (3 of 4)

ES074.0

Notes:

Checkpoint (4 of 4)

15. True or False: A PDSE cannot contain executable programs.
16. Name two utilities to allocate a zFS.
17. True or False: To allocate a zFS, you must specify in JCL, DSNTYPE=ZFS.
18. True or False: z/OS UNIX directory names and file names can be in uppercase, lowercase, or mixed case.
19. What is the data set organization of zFS?
 - a. VSAM
 - b. LDS
 - c. KSDS

© Copyright IBM Corporation 2011

Figure 5-47. Checkpoint (4 of 4)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Examine the record formats supported by z/OS
- Compare blocked and unblocked data sets
- Discuss system-determined block size
- Introduce the SMS-managed data sets
- Describe the data set organizations supported by z/OS
- Differentiate between PDS and PDSE data sets
- Explain how to access a z/OS UNIX System Services HFS data set and files
- Discuss VSAM data set creation through JCL

© Copyright IBM Corporation 2011

Figure 5-48. Unit summary

ES074.0

Notes:

Unit 6. Generation data groups

What this unit is about

There are instances in data processing when collections of data sets are related in some way. For example, a shop might dump a data set on a daily basis. This would create a number of data sets which are related by the fact that they are all dumps of the same data set. A generation data group (GDG) provides a convenient vehicle for establishing a relationship among these data sets. A generation data group is a group of related cataloged data sets.

What you should be able to do

After completing this unit, you should be able to:

- Describe when a generation data group is needed
- Code DD statements to utilize a generation data group in both single and multiple step jobs
- Discuss the differences between the relative data set name and the absolute data set name

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597

z/OS MVS JCL Reference

SA22-7598

z/OS MVS JCL User's Guide

Unit objectives

After completing this unit, you should be able to:

- Describe when a generation data group is needed
- Code DD statements to utilize a generation data group in both single and multiple step jobs
- Discuss the differences between the relative data set name and the absolute data set name

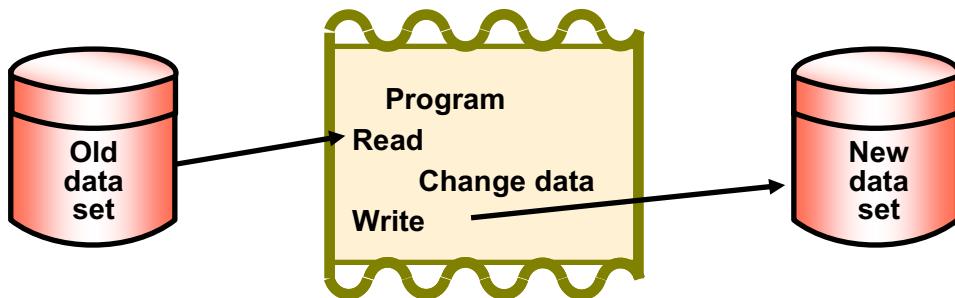
© Copyright IBM Corporation 2011

Figure 6-1. Unit objectives

ES074.0

Notes:

Generation data groups: The need



© Copyright IBM Corporation 2011

Figure 6-2. Generation data groups: The need

ES074.0

Notes:

A generation data group (GDG) consists of like-named data sets that are chronologically or functionally related. A data set in a GDG is called a generation.

Why use GDGs?

- **Case 1:** You attempt to catalog a new data set with the same name as an existing data set.
 - Catalog error! Name already in the catalog.
- **Case 2:** You keep the new data set with the same name.
 - JCL error! Duplicate name on VTOC.
- **Case 3:** You catalog or keep the data set with a different name.
 - JCL statements must be changed to the new names.

GDG: DSNAME specification

DSN=_____(+n)
(+n): ADD A NEW GENERATION
(+0): USE CURRENT GENERATION
(-n): USE AN OLD GENERATION
EXAMPLE:
DSN=NAME.GDG(+1)

DSN=_____.GxxxxVyy
GxxxxVyy:
xxxx: GENERATION NUMBER
yy: VERSION NUMBER
EXAMPLE:
DSN=NAME.GDG.G0052V00

© Copyright IBM Corporation 2011

Figure 6-3. GDG: DSNAME specification

ES074.0

Notes:

Generation data sets (generations) have two types of names:

Relative data set name:

The relative name is the most common form of GDG name used in JCL coding. The format is:

DSN=_____(+n) or DSN=NAME.GDG(-n)

Where n is the relative generation number of the GDG. For example:

- | | |
|----------------------------------|------------------|
| (+n) : Add a new generation | DSN=NAME.GDG(+1) |
| (0) : Use the current generation | DSN=NAME.GDG(0) |
| (-n) : Use an old generation | DSN=NAME.GDG(-1) |

Absolute data set name:

The absolute data set name is the true data set name you would see on the volume table of contents (VTOC). The format is:

```
DSN=_____.GxxxxVyy
      |       _____ Version Number
      |   _____ Generation Number
```

The generation number starts with G0001 for the first generation and is incremented by the value coded in the relative data set name. For example:

RELATIVE NAME	ABSOLUTE NAME	
NAME.GDG(+0)	NAME.GDG.G0003V00	CURRENT GENERATION
NAME.GDG(-1)	NAME.GDG.G0002V00	NEXT OLDEST GENERATION
NAME.GDG(-2)	NAME.GDG.G0001V00	OLDEST GENERATION

The smaller the relative number, the older the generation.

The V00 or version number is not used by IBM. The version number is set to V00 with the first generation. If a generation is damaged and needs to be replaced, an installation can code the absolute name with a new version number, to replace the damaged generation. For example:

```
//INDD DD DSN=NAME.GDG.G0052V00,DISP=OLD
//OUTDD DD DSN=NAME.GDG.G0052V01,DISP=(,CATLG)
```

GDG DSNAME: Base catalog entry

TEST.GDG	LIMIT(7)	NOEMPTY	NOSCRATCH	FOR (14) passwords
LIMIT	Maximum number of generations allowed for this GDG entry			
EMPTY	When limit exceeded, uncatalog all generations			
NOEMPTY	When limit exceeded, uncatalog oldest entry only			
SCRATCH	Delete the DSCB for any entry uncataloged			
NOSCRATCH	Do not delete the DSCB for any entry uncataloged			
FOR	Retention period			
TO	Expiration date			
OWNER	User information (up to eight characters)			
Passwords	Depend on protection mechanism used			

© Copyright IBM Corporation 2011

Figure 6-4. GDG DSNAME: Base catalog entry

ES074.0

Notes:

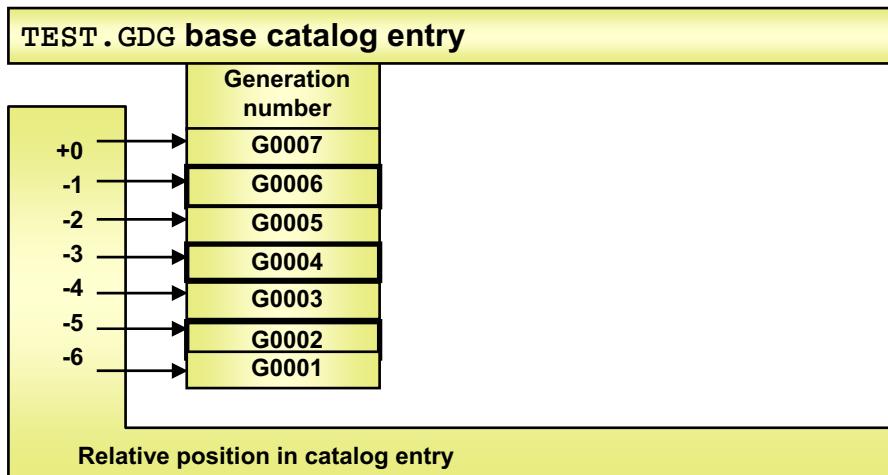
- Before generation data sets can be created, a GDG base catalog entry and model DSCB (or pattern DSCB) must be defined. If you define GDG on SMS managed volume, you do not define model DSCB because SMS does not support it.
- The IDCAMS utility is used to create the GDG base catalog entry. Five key IDCAMS control statement parameters which are used in creating the entry are:
 - **LIMIT**: Maximum number of generations allowed for this GDG entry
 - **EMPTY**: When limit is exceeded, uncatalog all generations
 - **NOEMPTY**: When limit is exceeded, uncatalog oldest entry only
 - **SCRATCH**: Delete any uncataloged generation
 - **NOSCRATCH**: Do not delete any uncataloged generation
- The model DSCB ensures that the same DCB and expiration date information is used to create all generations. This ensures greater consistency among generations.

- The model DSCB must be allocated with zero spaces, and it cannot be cataloged. The model DSCB must also lie on the same volume where the base catalog entry is cataloged.
- A sample IDCAMS job stream which can be used to create *both* the catalog entry and build a model DSCB is shown next:

```
//DEFGDG    JOB   ...
//STEP1      EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//DSCB       DD  DSN=TEST.GDG,DISP=(,KEEP),UNIT=SYSDA,
//                  SPACE=(TRK,0),VOL=SER=EDPAK1,
//                  DCB=(RECFM=FB,LRECL=100)
//SYSIN      DD  *
      DEFINE GENERATIONDATAGROUP -
      (NAME(TEST.GDG) -
      NOEMPTY -
      NOSCRATCH -
      LIMIT(7))
```

- You do not need to create a model data set label if any of the following is true:
 - You can refer to a cataloged data set with attributes identical to those you want or to an existing model data set label for which you can supply overriding attributes.
 - The DCB attributes are supplied by the specified or selected data class.
 - The DCB attributes are specified on the DD statement or on the DCB in the program that creates the data set

GDG DSNAME: Catalog entry


Examples:

//DD1 DD DSN=TEST.GDG (+0),DISP=OLD	Locates current (top) entry
//DD2 DD DSN=TEST.GDG (-6),DISP=OLD	Locates oldest entry

© Copyright IBM Corporation 2011

Figure 6-5. GDG DSNAME: Catalog entry

ES074.0

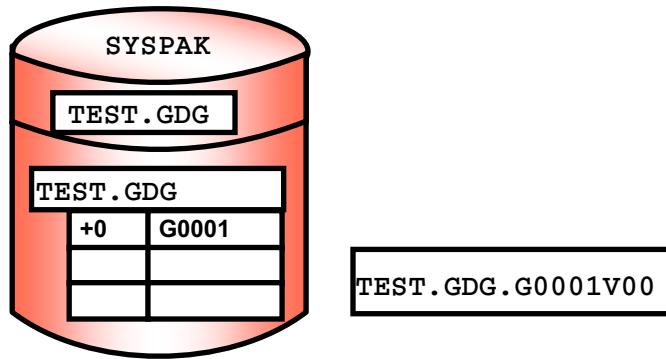
Notes:

This is a view of a GDG catalog entry.

Notice that the catalog entry operates like a push-down stack.

GDG example: First generation

```
//EXAMPLE      JOB      378,SMITH,CLASS=T
//STEP1        EXEC    PGM=USERPGM1
//FIRST        DD      DSN=TEST.GDG(+1),DISP=(,CATLG),
//                           DATACLAS=DFIX,MGMTCLAS=WKLY
//INPUT        DD      DSN=INITIAL.DATA,DISP=OLD
```



© Copyright IBM Corporation 2011

Figure 6-6. GDG example: First generation

ES074.0

Notes:

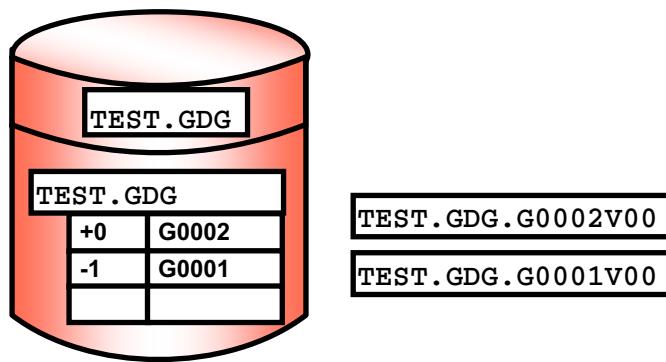
Assume that the GDG generations are SMS-managed. (This assumption is not critical to the job processing. The *submitter* of this job will not observe any processing differences in SMS and non-SMS environments; however, the system processes the job differently in the two environments.)

In the example in the visual, the first generation of the generation data group named TEST.GDG is created.

Note that DISP=(,CATLG) is required when creating a new generation.

GDG example: Second generation

```
//EXAMPLE2      JOB    378, SMITH, CLASS=G
//STEPX        EXEC   PGM=MAINLINE
//GGIN         DD     DSN=TEST.GDG (+0), DISP=OLD
//GOUT         DD     DSN=TEST.GDG (+1), DISP=(NEW, CATLG),
//                           DATACLAS=DFIX, MGMTCLAS=WKLY
```



© Copyright IBM Corporation 2011

Figure 6-7. GDG example: Second generation

ES074.0

Notes:

In the example in the visual, the second generation of the generation data group named TEST.GDG is created.

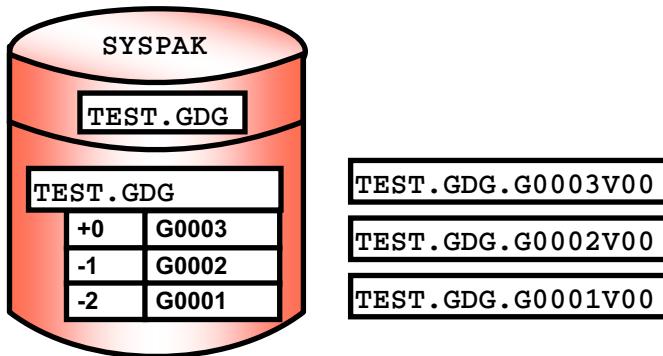
The current generation (0) is used as input. You are not required to code the plus sign with the zero (+0); a zero without the plus sign (0) is acceptable and commonly used.

A new generation (+1) is created.

Notice that the catalog entry operates like a push-down stack.

GDG example: Third generation

```
//EXAMPLE2      JOB    378, SMITH, CLASS=G
//STEPX        EXEC   PGM=MAINLINE
//GGIN         DD     DSN=TEST.GDG (+0), DISP=OLD
//GGOOUT        DD     DSN=TEST.GDG (+1), DISP=(NEW, CATLG),
//                           DATACLAS=DFIX, MGMTCLAS=WKLY
```



© Copyright IBM Corporation 2011

Figure 6-8. GDG example: Third generation

ES074.0

Notes:

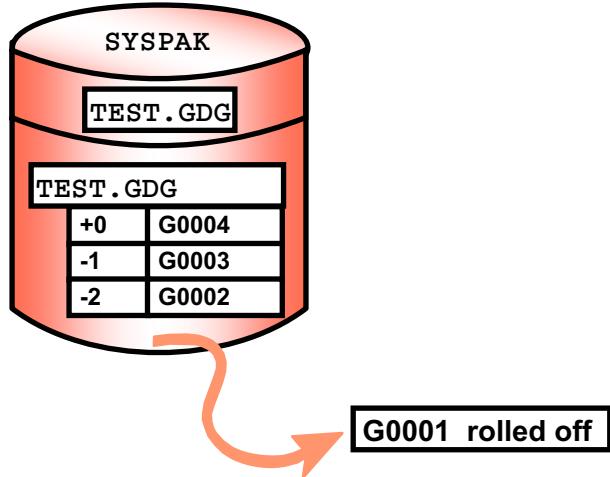
In the example in the visual, the third generation of the generation data group named TEST.GDG is created.

The current generation (0) is used as input.

A new generation (+1) is created.

Notice that the catalog entry operates like a push-down stack.

GDG example: Limit exceeded



```
//EXAMPLE2   JOB    378,SMITH,CLASS=G
//STEPX          EXEC    PGM=MAINLINE
//GDGIN          DD      DSN=TEST.GDG(+0),DISP=OLD
//GDGOUT         DD      DSN=TEST.GDG(+1),DISP=(NEW,CATLG),
//                  DATACLAS=DFIX, MGMTCLAS=WKLY
```

© Copyright IBM Corporation 2011

Figure 6-9. GDG example: Limit exceeded

ES074.0

Notes:

In the above example, the fourth generation of the generation data group named TEST.GDG is created.

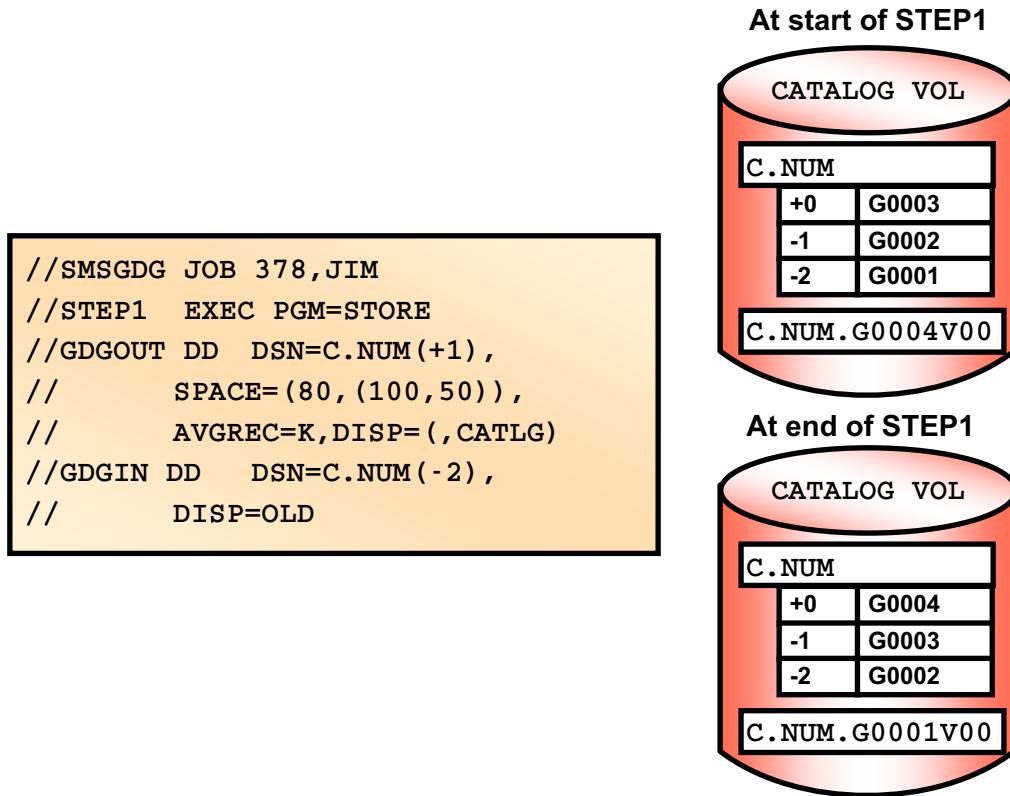
The current generation (0) is used as input.

A new generation (+1) is created.

This causes the GDG limit to be exceeded.

Thus the first generation, TEST.GDG.G0001V00, will be removed from the list of cataloged entries. Since NOSCRATCH was specified, this generation will be kept. Since the generations are SMS-managed, this generation will remain in the catalog. If the generations had not been SMS-managed, the generation would have been uncataloged.

GDG ROLLIN and ROLLOFF



© Copyright IBM Corporation 2011

Figure 6-10. GDG ROLLIN and ROLLOFF

ES074.0

Notes:

GDG processing works slightly differently in non-SMS and SMS environments. These changes are illustrated by an example.

Assume a GDG with a base name, C.NUM, was defined with attributes LIMIT(3), NOEMPTY, and NOSCRATCH, and assume three generations are cataloged. Suppose someone submits the job illustrated in the visual.

In non-SMS environments, cataloging is done at step termination. Thus, the first generation (DSN=C.NUM.G0001V00) is available when the program STORE references the DD statement GDGIN. At step termination, the first generation is uncataloged (but not scratched), and the fourth generation (DSN=C.NUM.G0004V00) is cataloged.

In SMS environments, cataloging is done at step initiation. If changes had not been made to GDG processing, the first generation would have been uncataloged at the beginning of the step (to make room for the fourth generation). Hence, STORE would have been unable to access the first generation, an inconsistent situation.

To insure consistency between environments, the fourth generation will be cataloged in a special entry at step initiation. This is called a *deferred roll-in state*.

At step termination, the first generation will be removed from the list of catalogued generations, and the fourth generation will be rolled in (that is, catalogued). Since the GDG was defined with NOSCRATCH, the first generation will be kept; however, all SMS-managed data sets are to be catalogued, so the first generation will be catalogued in a special entry. This is called a *rolled-off* or *rolled-out state*. It can still be referenced by specifying the absolute name (DSN = C.NUM.G0001V00).

GDG in a multistep job

```

//STEP1 EXEC PGM=ONE
//INBIG      DD           DSN=A.AIR,DISP=OLD
//OUT1       DD           DSN=A,AIR(+1),DISP=(,CATLG,DELETE) ....
      ....
//STEP2 EXEC PGM=TWO
//IN2        DD           DSN=PROD,DATA,DISP=OLD
//OUT2       DD           DSN=A.AIR(+2),DISP=(,CATLG,DELETE) ....
      ....
//STEP3 EXEC PGM=THREE
//IN3        DD           DSN=A.AIR(+1),DISP=OLD
//                      DD           DSN=A.AIR(+2),DISP=OLD

```

© Copyright IBM Corporation 2011

Figure 6-11. GDG in a multistep job

ES074.0

Notes:

Here are the explanations for the use of GDG in the multistep job in the visual:

- STEP1
 - A GDGALL request (DSN=A.AIR) is IBM's terminology for automatic concatenation of *all* generations currently in the catalog.
- STEP3
 - Notice that the (+1) generation is now OLD. Even though that generation was cataloged in STEP1 and the catalog pushed down, you do not adjust the relative generation numbers. The system does that internally.
 - As additional generations are specified in this step, increment the relative generation number. An increment of 1 is shown, but any increment can be used.



Note

Relative GDG numbers are not updated until end of job; therefore, the relative numbers are held constant throughout a single job.

Use `DISP=OLD` when processing a GDG to prevent other jobs (on the same system) from processing the same GDG at the same time.

Checkpoint (1 of 2)

1. Which of the following have valid JCL GDG references to the current generation?
 - a. //GDGIN DD DSN=TEST.GDG(0),DISP=OLD
 - b. //GDGIN DD DSN=TEST.GDG(+0),DISP=OLD
 - c. Both //GDGIN DD DSN=TEST.GDG(0),DISP=OLD and //GDGIN DD DSN=TEST.GDG(+0),DISP=OLD.

2. Relative GDG numbers are not updated until end of:
 - a. Job
 - b. Step

3. True or False: Once the job ends either normally or abnormally, the most recent entry in the catalog is now the +0 generation.

© Copyright IBM Corporation 2011

Figure 6-12. Checkpoint

ES074.0

Notes:

Checkpoint (2 of 2)

4. If G0001V00 is the current generation and a new generation is being created with the JCL coded DSN=NAME.GDG (+3) , what would the new absolute name be?

5. Match the following:
 - NAME.GDG(-1)
 - OLDEST GENERATION
 - CURRENT GENERATION
 - NAME.GDG(+0)
 - NAME.GDG.G0003V00
 - NAME.GDG.G0002V00
 - NAME.GDG(-2)
 - NEXT OLDEST GENERATION
 - NAME.GDG.G0001V00

© Copyright IBM Corporation 2011

Figure 6-13. Checkpoint (2 of 2)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Describe when a generation data group is needed
- Code DD statements to utilize a generation data group in both single and multiple step jobs
- Discuss the differences between the relative data set name and the absolute data set name

© Copyright IBM Corporation 2011

Figure 6-14. Unit summary

ES074.0

Notes:

Unit 7. Procedures

What this unit is about

Some jobs contain essentially the same JCL statements and are repeatedly executed by one or more users. These JCL statements can be placed in a procedure. Users can then execute the procedure and save the time and errors that arise from coding the same JCL over and over again. This topic investigates the rules for coding and using procedures.

What you should be able to do

After completing this unit, you should be able to:

- Describe a procedure
- Differentiate between a cataloged procedure and instream procedure
- Describe procedure modifications through overriding, adding, or nullifying parameters
- Use symbolic parameters to modify procedures
- Compare the `PROC` and `EXEC` statements for supplying symbolic parameters

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597

z/OS MVS JCL Reference

Unit objectives

After completing this unit, you should be able to:

- Describe a procedure
- Differentiate between a cataloged procedure and instream procedure
- Describe procedure modifications through overriding, adding, or nullifying parameters
- Use symbolic parameters to modify procedures
- Compare the `PROC` and `EXEC` statements for supplying symbolic parameters

© Copyright IBM Corporation 2011

Figure 7-1. Unit objectives

ES074.0

Notes:

Procedures (1 of 2)

- Consist of multiple executable steps
- Accessed by the EXEC statement
- Supported in every version and release of z/OS

© Copyright IBM Corporation 2011

Figure 7-2. Procedures (1 of 2)

ES074.0

Notes:

A *procedure* is a named collection of JCL stored in a data set.

- For jobs that you run frequently or jobs that use the same JCL, pre-code job control statements into procedures.
- Procedures consist of one or more complete steps.
- Every procedure must be given a name.
- Procedures are invoked through the EXEC statement.
- Three benefits of using procedures are:
 - It saves time by reducing the time required to code JCL.
 - It saves library storage by eliminating duplicate JCL.
 - It reduces JCL errors by providing access to debugged JCL.

Example:

```
//STEP    EXEC PROC=procedure name  
or  
//STEP    EXEC procedure name
```

Procedures (2 of 2)

- To execute, code:

```
//STEP EXEC PROC=Procedure Name
```

- Procedures can be nested up to 15 levels

Two types

Instream	Included in a job prior to attempt to execute it
Cataloged	Stored in PROCLIB so it can be omitted from job

© Copyright IBM Corporation 2011

Figure 7-3. Procedures (2 of 2)

ES074.0

Notes:

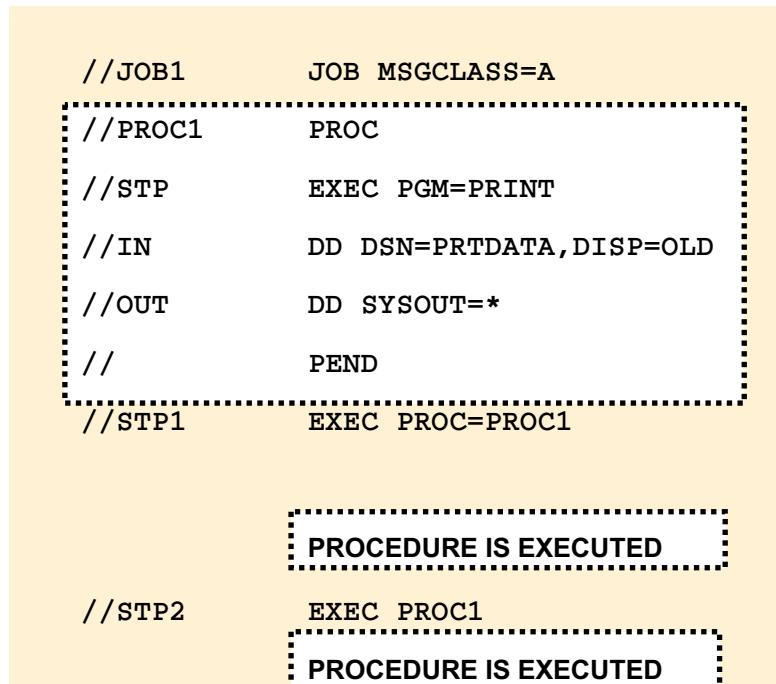
There are two types of procedures:

- When you place a procedure in the job input stream, it is called an *instream* procedure.
 - Primarily used to test procedures.
 - Reside in the job input stream and can be called only from that job stream.
- A procedure cataloged in a library is called a *cataloged procedure*.
 - Can be called when needed.
 - Resides in a procedure library.

Procedures cannot contain:

- JOB statements
- DD * statements
- DD DATA statements
- Delimiter statements ('/*')
- Null statements ('//')
- Non-JCL statements (for example, JES or utility control statements)

Instream procedure



© Copyright IBM Corporation 2011

Figure 7-4. Instream procedure

ES074.0

Notes:

Place an instream procedure in the input stream:

- After any `JOB` statement
- Before any `EXEC` statement that calls it

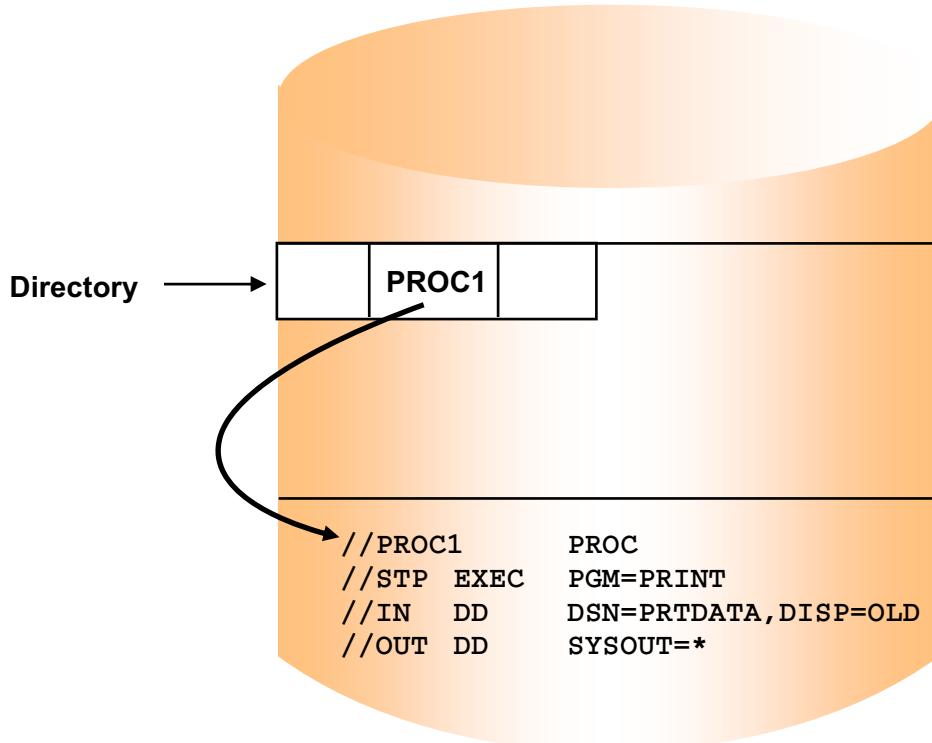
An instream procedure:

- Must begin with a `PROC` statement
- Must end with a `PEND` statement
- Is called by an `EXEC` statement using the procedure name
- Must be resubmitted each time the job is executed

Example:

```
//JOB1    JOB   MSGCLASS=A  
//PROC1   PROC  
//STEP1   EXEC  PGM=PRINT  
//IN      DD DSN=PRTDATA,DISP=OLD  
//OUT     DD SYSOUT=*  
//          PEND  
//ST1     EXEC  PROC1  
//ST2     EXEC  PGM=TEST1  
//INDD    DD DSN=INDATA,DISP=OLD  
//OUTDD   DD DSN=PRTDATA,DISP=OLD
```

Cataloging a procedure



© Copyright IBM Corporation 2011

Figure 7-5. Cataloging a procedure

ES074.0

Notes:

After an instream procedure has been tested during execution, it can be cataloged. To use it, call it with an `EXEC` statement. Such a procedure is sometimes known as a *cataloged procedure*. Do not confuse a cataloged procedure with a cataloged data set. A cataloged procedure is a named collection of JCL stored in a data set, and a cataloged data set is a data set whose name is recorded by the system.

- A procedure is said to be cataloged when it is placed in a procedure library (proclib).
- A proclib is a PDS or PDSE, and the member name is the procedure name coded on the calling `EXEC` statement.
- A procedure can be cataloged by placing it in one of three types of proclibs:
 - `SYS1.PROCLIB`: IBM-supplied system procedure library.
 - System proclibs: defined by an installation.
 - User-defined proclib.
- Use the `IEBUPDTE` or `IEBCOPY` utility or ISPF/PDF to add a procedure to a proclib or modify a procedure. (`IEBUPDTE` is described in the next topic.)

- A PROC statement can be included in a cataloged procedure, but it is not required.
- **Example:**

```
//PROC1 PROC  
//MYSORT EXEC PGM=SORT  
//SORTIN DD DISP=SHR,DSN=&SORTDSN  
//SORTOUT DD SYSOUT=*  
//SYSOUT DD SYSOUT=*  
// PEND
```

```
//JOB1 JOB ....  
//      EXEC PROC=PROC1
```

Or

```
//JOB1 JOB  
//      EXEC PROC1
```

Procedure modification

- Two methods of modifying procedures:
 - Symbolic parameters
 - Overriding, adding, or nullifying parameters
- Either method can be used to modify:
 - Instream and cataloged procedures
 - EXEC, DD, or OUTPUT JCL statements

© Copyright IBM Corporation 2011

Figure 7-6. Procedure modification

ES074.0

Notes:

It is most efficient to design a procedure so that you can use symbolic parameters to modify it.

A symbolic parameter can stand as a symbol for any information in the parameter field of a statement.

Alternately, you can modify an instream or cataloged procedure by:

- Overriding, nullifying, or adding EXEC statement parameters.
- Overriding, nullifying, or adding parameters to DD or OUTPUT JCL statements.
- Adding DD or OUTPUT JCL statements.

To make modifications to a procedure, you must know STEPNAMEs and DDNAMEs. You must also know what step the DDNAME is in.

Modifying EXEC statements

- EXEC parameters can be overridden, added, or nullified.
- Code MODS on EXEC statement that invokes PROC.
- Code MODS for one PROC step before the next PROC step.

© Copyright IBM Corporation 2011

Figure 7-7. Modifying EXEC statements

ES074.0

Notes:

Note the following principles that apply when modifying EXEC statements:

- Existing EXEC statement parameters can be overridden or nullified.
- New EXEC statement parameters can be added.
- All modifications to an EXEC statement in a procedure must be specified on the EXEC statement that invokes the procedure.
 - If the parameter to be modified is found in the procedure, the value is overridden.
 - If the parameter to be modified is not found, the parameter is added.
- Modifications apply to one level of nesting only.

Modify EXEC statement syntax

//STEP1 EXEC=_, KEYWORD. PROCSTEPNAME=VALUE

Parameter
to be
modified

Which
step

Procedure 'P1'

```
//STEP1 EXEC      PGM=PAYROLL,TIME=(2,30),ACCT=1876
//STEP2 EXEC      PGM=PRINT,TIME=(4,30)
```

Job stream

```
//XY2   JOB
//STEPA EXEC      PROC=P1,TIME.STEP1=(1,10),
// ACC.STEP1=,PARM.STEP2=TOP
```

Resulting JCL

```
//STEP1 EXEC      PGM=PAYROLL,TIME=(1,10)
//STEP2 EXEC      PGM=PRINT,TIME=(4,30),PARM=TOP
```

© Copyright IBM Corporation 2011

Figure 7-8. Modify EXEC statement syntax

ES074.0

Notes:

Observe the following principles regarding syntax when modifying EXEC statements:

- To override a parameter in a step, code `PARAMETER.PROCSTEPNAME=value` on the invoking EXEC statement.
- 'PGM=' cannot be overridden.

Example :

```
// EXEC PAY,TIME.STEP1=(,10)
    |   |     | Value
    |   |     | Procedure step name
    |     | Parameter to be overridden
```

- To nullify a parameter in a step, code `PARAMETER.PROCSTEPNAME=` without specifying a value after the equals sign (=).

Example:

```
// EXEC PAY,ACCT.STEP1=
      | _____ Procedure step name
      | _____ Parameter to be nullified
```

- To modify a parameter in every step, code the parameter as usual.

Example:

```
// EXEC PAY,ACCT=1996
      | _____ Parameter to be modified in every step
```

- On the EXEC statement that calls the procedure, code all parameters *without* step names *before* coding parameters *with* step names.
- Code all EXEC statement parameters in the same order the steps are in. That is, code parameters for one step before coding those for succeeding steps.

Example:

```
// EXEC PAY,ACCT=1996,PARM.STEP1=PAYOUT,
// REGION.STEP1=100K,REGION.STEP2=400K,TIME.STEP3=(1,30)
```

Modifying DD statements

- DD parameters can be overridden, added, or nullified.
- Code modifications on DD statements following EXEC statement.
- Code modifications for one PROC step before the next PROC step.
- Nullify a keyword PARM by coding KEYWORD=.

© Copyright IBM Corporation 2011

Figure 7-9. Modifying DD statements

ES074.0

Notes:

DD statement parameters can be overridden, added, or nullified in any procedure step.

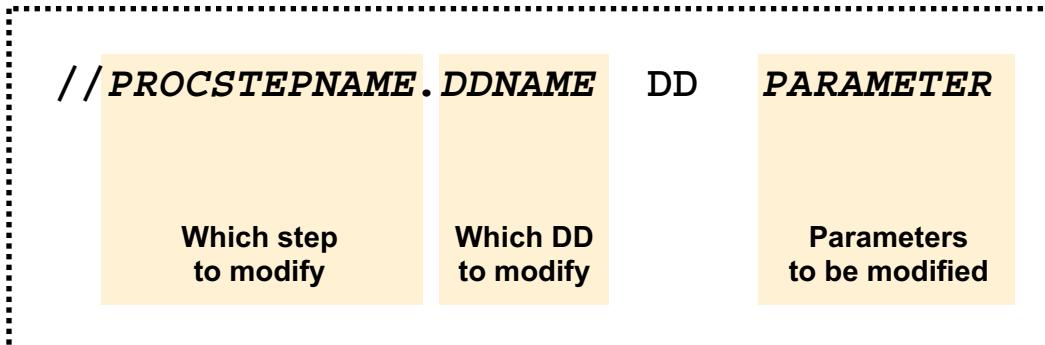
Code the modifying DD statement following the EXEC statement that calls the procedure.

Modifying DD statements can be coded in any order in a step, as long as the ddname has the form *PROCSTEPNAME.DDNAME*.

DD statements can be added with the modifying DD statements in any order in a step.

Up to 15 levels of nesting are allowed in z/OS but modifications apply to one level of nesting only.

Modify DD statement syntax



© Copyright IBM Corporation 2011

Figure 7-10. Modify DD statement syntax

ES074.0

Notes:

To modify a DD statement in a procedure:

- Code a modifying DD statement following the EXEC statement that calls the procedure.
- In the DDNAME field, code the PROCSTEPNAME.DDNAME DD parameter to be modified.

For example:

```
//      EXEC PAY
//STEP1.DD1  DD DSN=THESE
```

			Parameter to be modified
		DD name	
	Procedure step name		

Overriding DD statements can be placed in any order in a step. DD statements can be added in any order in a step.

DD statement modifications need not be complete statements; that is, you code only the parameters you want changed or nullified.

DD statement additions must be complete DD statements.

Modify DD: Example (1 of 2)

```

Procedure 'P1'
//S1      EXEC PGM=PAYROLL
//A       DD   DSN=INPUT,DISP=OLD
//B       DD   DSN=OUTPUT,DISP=(,CATLG,DELETE),UNIT=3350,
//                  SPACE=(CYL,(20,5)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=320)
//C       DD   DSN=SALARY,DISP=OLD,UNIT=3390,VOL=SER=PAK10
//S2      EXEC PGM=PRINT
//A       DD   DSN=OUTPUT,DISP=(OLD,DELETE),UNIT=3350,VOL=SER=PAK08
//B       DD   SYSOUT=*

Job stream
//JOB1  JOB  MSGCLASS=A
//FS     EXEC P1
//S1.A  DD   DISP=(OLD,DELETE,KEEP)
//S1.B  DD   UNIT=3390,SPACE=(4096,(100,2000)),DCB=(BLKSIZE=800)
//S1.D  DD   *
      DATA
//S2.A  DD   UNIT=,VOL=SER=,DISP=OLD

```

© Copyright IBM Corporation 2011

Figure 7-11. Modify DD: Example (1 of 2)

ES074.0

Notes:

Shown first is the procedure P1.

Shown second is a job stream that invokes the procedure P1 with DD statement modifications and a DD statement addition.

- In the procedure step S1, DD statements A and B will be modified, and DD statement D will be added.
- In the procedure step S2, DD statement A will be modified.
- All modifications for procedure step S1 are coded before the modifications for procedure step S2.

Modify DD: Example (2 of 2)

```
//JOB1 JOB MSGCLASS=A
//S1    EXEC PGM=PAYROLL
//A     DD DSN=INPUT,DISP=(OLD,DELETE,KEEP)
//B     DD DSN=OUTPUT,DISP=(NEW,CATLG),UNIT=3390,VOL=SER=PAK03,
//          SPACE=(4096,(100,20)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//C     DD DSN=SALARY,DISP=OLD,UNIT=3390,VOL=SER=PAK10
//D     DD *
*
DATA
//S2    EXEC PGM=PRINT
//A     DD DSN=OUTPUT,DISP=OLD
//B     DD SYSOUT=*
```

© Copyright IBM Corporation 2011

Figure 7-12. Modify DD: Example (2 of 2)

ES074.0

Notes:

The resulting JCL will appear to contain more JCL-like statements than you submitted. The resulting JCL will show both the JCL you submitted and the JCL in the invoked proc. The // in columns 1 and 2 of the *invoked* proc does not appear in the same form as the submitted JCL. There are two cases.

- If the proc is cataloged, the // in the proc will be changed to:
 - xx if the statement was not overridden.
 - x/ if at least one parameter on the statement was overridden. (The overriding statement precedes the overridden statement.)
 - xx* if the statement was not a comment but was changed to a comment.
- If the proc is instream, the // in the proc will be changed to:
 - ++ if the statement was not overridden.
 - +/ if at least one parameter on the statement was overridden. (The overriding statement precedes the overridden statement.)
 - ++* if the statement was not a comment but was changed to a comment.

Modifying procedures with symbolic parameters

//A DD VOL=SER=&V, . . .

- Assign values to symbolics through:
 - PROC: Assigns PROC defaults
 - EXEC: Overrides PROC defaults
 - SET: Also assigns values in non-PROC JCL

```
//S { PROC  
EXEC P1,  
SET } V=VOL001
```

© Copyright IBM Corporation 2011

Figure 7-13. Modifying procedures with symbolic parameters

ES074.0

Notes:

A symbolic parameter is used to allow a JCL parameter to be easily changed or to be specified at execution time. Symbolic parameters are used in instream and cataloged procedures.

Any parameter, subparameter, or value in a procedure that can vary each time the procedure is called is a good symbolic parameter candidate.

A symbolic parameter consists of an ampersand (&) followed by a name which is one to eight alphanumeric or national characters.

Values assigned to the symbolic parameter through the PROC statement are the default values.

A symbolic parameter value is assigned by coding the symbolic parameter, without the ampersand (&), and its value. These can appear in any order on the statement. Each appearance of the symbolic in the procedure will have this value assigned.

In the following example, the symbolic parameter UN will have the default value of 3390 and OUT will have the default value of OUTPUT in the procedure named TESTPROC.

```
//TESTPROC    PROC  UN=3390,OUT=OUTPUT  
...  
//PSTEP4      DD   DSN=&OUT,DISP=(,CATLG),UNIT=&UN,  
...
```

The procedure default values can be overridden by coding the symbolic parameter value on the EXEC statement that invokes the procedure.

This example shows how to override the default value for the symbolic parameter UN. In the procedure TESTPROC, UN has default value of 3390. When the PROC is invoked, the override on the EXEC statement changes the value of UN to 3380.

```
//TESTPROC    PROC  UN=3390,OUT=OUTPUT  
...  
//PSTEP4      DD   DSN=&OUT,DISP=(,CATLG),UNIT=&UN,  
...  
//STEP1      EXEC  TESTPROC,UN=3380  
...
```

The SET statement can also be used to assign values to symbolic parameters. It offers greater flexibility than do the PROC or EXEC statements. SET will be discussed later in the class.

Symbolic parameters: Example 1 (1 of 2)

<pre>//JOB1 JOB MSGCLASS=A //S1 EXEC TESTPROC,ACT=7834,D2=CATLG,UN=3390,SR=PAK03, // U=4096,P='100,',S=20,BLK=800 //S1.D DD * DATA</pre>	<div style="border: 1px solid black; padding: 5px; text-align: center;">Job stream</div> <div style="border: 1px solid black; padding: 5px; text-align: center;">Procedure 'TESTPROC'</div> <pre>//TESTPROC PROC A=PAYROLL,ACT=1209,DP=OLD,D2=KEEP,UN=3350,SR=PAK08, // BLK=320,U=CYL,S=10,P=20 //S1 EXEC PGM=&A,ACCT=&ACT,TIME=(1,30) //A DD DSN=INPUT,DISP=&DP //B DD DSN=OUTPUT,DISP=(NEW,&D2),UNIT=&UN,VOL=SER=&SR, // SPACE=(&U,(&P&S)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=&BLK) //C DD DSN=SALARY,DISP=OLD,UNIT=3390,VOL=SER=PAK10 //S2 EXEC PGM=PRINT //A DD DSN=OUTPUT,DISP=(OLD,DELETE),UNIT=&UN,VOL=SER=&SR //B DD SYSOUT=*</pre>
---	--

© Copyright IBM Corporation 2011

Figure 7-14. Symbolic parameters: Example 1 (1 of 2)

ES074.0

Notes:

Default values are assigned to each of the symbolic parameters on the `PROC` statement at the beginning of the procedure.

Symbolic parameter values that override the default values are coded on the `EXEC` statement that invokes the procedure.

Note the `P='100,', S=20` on the `EXEC` statement that invokes `TESTPROC`. Another `P, S` coding combination which would give the same result is `P=100, S='1,20'`.

How could you modify the coding of the two symbolic parameters `&P` and `&S` in the `SPACE` parameter of the procedure to enable the submitter to code `P=100, S=20` on the invoking `EXEC` statement? Would you make this change in the coding of the two symbolic parameters in the `SPACE` parameter?

Symbolic parameters: Example 1 (2 of 2)

Resulting JCL

```
//JOB1  JOB  MSGCLASS=A
//S1      EXEC PGM=PAYROLL,ACCT=7834,TIME=(1,30)
//A       DD   DSN=INPUT,DISP=OLD
//B       DD   DSN=OUTPUT,DISP=(NEW,CATLG),UNIT=3390,VOL=SER=PAK03,
//           SPACE=(4096,(100,20)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//C       DD   DSN=SALARY,DISP=OLD,UNIT=3390,VOL=SER=PAK10
//D       DD   *
          DATA
//S2      EXEC PGM=PRINT
//A       DD   DSN=OUTPUT,DISP=(OLD,DELETE),UNIT=3390,VOL=SER=PAK03
//B       DD   SYSOUT=*
```

© Copyright IBM Corporation 2011

Figure 7-15. Symbolic parameters: Example 1 (2 of 2)

ES074.0

Notes:

Note the value used for each symbolic parameter.

Procedure ABCPROC

PROC exercise

© Copyright IBM Corporation 2011

Figure 7-16. Procedure ABCPROC

ES074.0

Notes:

Classroom exercise:

Use the procedure on the following page to complete the following steps:

Code the invoking JCL to:

- **STEP1:**

- Add a REGION of 256 KB.
- Remove BLKSIZE from the TRANS file.
- Catalog the data set TRANS and put it on the 3390 disk EDPAK8.

- **STEP2:**

- Remove the ACCT parameter.

- **STEP3:**

- Add a PARM parameter of JANUARY.

Procedure ABCPROC

```
//ABCPROC      PROC   UN=3380
//STEP1        EXEC PGM=SORT
//SORTIN        DD DSN=TIMERCDS,DISP=SHR
//SORTOUT       DD DSN=TRANS,DISP=(,PASS),
//                           SPACE=(TRK,(2,1)),
//                           UNIT=&UN,VOL=SER=EDPAK2,
//                           DCB=(LRECL=80,BLKSIZE=8000,RECFM=FB)
//SYSOUT        DD SYSOUT=*
//SYSIN         DD DSN=TRANS.SORT.CNTL,DISP=SHR
//STEP2        EXEC PGM=PAYROLL,ACCT=1492
//PAYIN         DD DSN=MSTRIN,DISP=SHR
//PAYTRANS      DD DSN=TRANS,DISP=SHR
//PAYOUT        DD DSN=MSTROUT,DISP=(,CATLG),SPACE=(TRK,(50,5)),
//                           UNIT=SYSDA,VOL=SER=EDPAK3,
//                           DCB=(LRECL=100,BLKSIZE=8000,RECFM=FB)
//CHECKS        DD SYSOUT=(A,CHKS)
//REPORT        DD SYSOUT=*
//STEP3        EXEC PGM=PAYACCT
//MSTR          DD DSN=MSTROUT,DISP=SHR
//REPORT1       DD SYSOUT=*
//REPORT2       DD SYSOUT=*
```

Checkpoint (1 of 3)

1. Which statement below is a valid statement to invoke a PROC?
 - a. //STEP EXEC PROC=*procedure name*
 - b. //STEP EXEC *procedure name*
 - c. Both //STEP EXEC PROC=*procedure name* and //STEP EXEC *procedure name*.

2. What are the two types of procedures?

3. Give a data set name where a cataloged procedure can be stored.

© Copyright IBM Corporation 2011

Figure 7-17. Checkpoint (1 of 3)

ES074.0

Notes:

Checkpoint (2 of 3)

4. True or False: Procedures can contain in stream SYSIN
`(//SYSIN DD *).`
5. True or False: DD statement parameters can be overridden, added, or nullified in any procedure step.
6. What are the two statements required in an instream procedure (beginning and end)?
7. True or False: DD statements which can be added with the modifying DD statements must appear last after all existing DD in a step.

© Copyright IBM Corporation 2011

Figure 7-18. Checkpoint (2 of 3)

ES074.0

Notes:

Checkpoint (3 of 3)

8. True or False: A symbolic parameter consists of an && followed by a name which is one to eight alphanumeric or national characters.
9. True or False: Default values must be assigned to each of the symbolic parameters on the PROC statement at the beginning of the procedure.
10. Assuming a proc defined as //PROCSMP PROC P1=, P2=, what is the valid way to call this proc?
 - a. //STEP1 EXEC PROCSMP, P1=PARM1, P2=PARM2
 - b. //STEP1 EXEC PROCSMP, P2=PARM1, P1=PARM2
 - c. Both //STEP1 EXEC PROCSMP, P1=PARM1, P2=PARM2 and //STEP1 EXEC PROCSMP, P2=PARM1, P1=PARM2.

© Copyright IBM Corporation 2011

Figure 7-19. Checkpoint (3 of 3)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Describe a procedure
- Differentiate between a cataloged procedure and instream procedure
- Describe procedure modifications through overriding, adding, or nullifying parameters
- Use symbolic parameters to modify procedures
- Compare the `PROC` and `EXEC` statements for supplying symbolic parameters

© Copyright IBM Corporation 2011

Figure 7-20. Unit summary

ES074.0

Notes:

Unit 8. More about utilities

What this unit is about

Certain routine tasks involving data set organization and maintenance arise frequently in a data processing environment. It would take substantial time for a programmer to repeatedly write programs to handle these tasks. Therefore, many IBM-supplied utilities are available to process these commonly occurring tasks. This unit introduces additional utility routines.

What you should be able to do

After completing this unit, you should be able to:

- Look at examples of more utility programs
 - IEBGENER
 - IEBCOPY
 - IEHLIST
 - IEHPROGM
 - IEBUPDTE
 - BPXBATCH
- Use the *DFSMS/dfp Utilities manual* for reference

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597 z/OS MVS JCL Reference

SC26-7414 DFSMS/dfp Utilities

Unit objectives

After completing this unit, you should be able to:

- Look at examples of more utility programs:
 - IEBGENER
 - IEBCOPY
 - IEHLIST
 - IEHPROGM
 - IEBUPDTE
 - BPXBATCH
- Use the *DFSMS/dfp Utilities* manual for reference

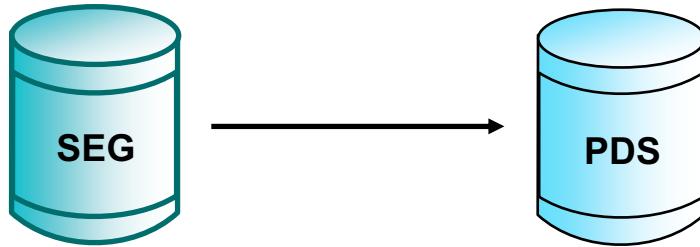
© Copyright IBM Corporation 2011

Figure 8-1. Unit objectives

ES074.0

Notes:

IEBGENER copy



```
//COPY      EXEC      PGM=IEBGENER
//SYSPRINT DD        SYSOUT=*
//SYSUT1   DD        DSN=SEQ.DATA,DISP=SHR
//SYSUT2   DD        DSN=PDSE.DATA(MEM),DISP=(,CATLG),
//                  RECFM=FB,LRECL=80,UNIT=SYSDA,SPACE=(TRK,(5,1,5))
//SYSIN    DD        DUMMY
```

© Copyright IBM Corporation 2011

Figure 8-2. IEBGENER copy

ES074.0

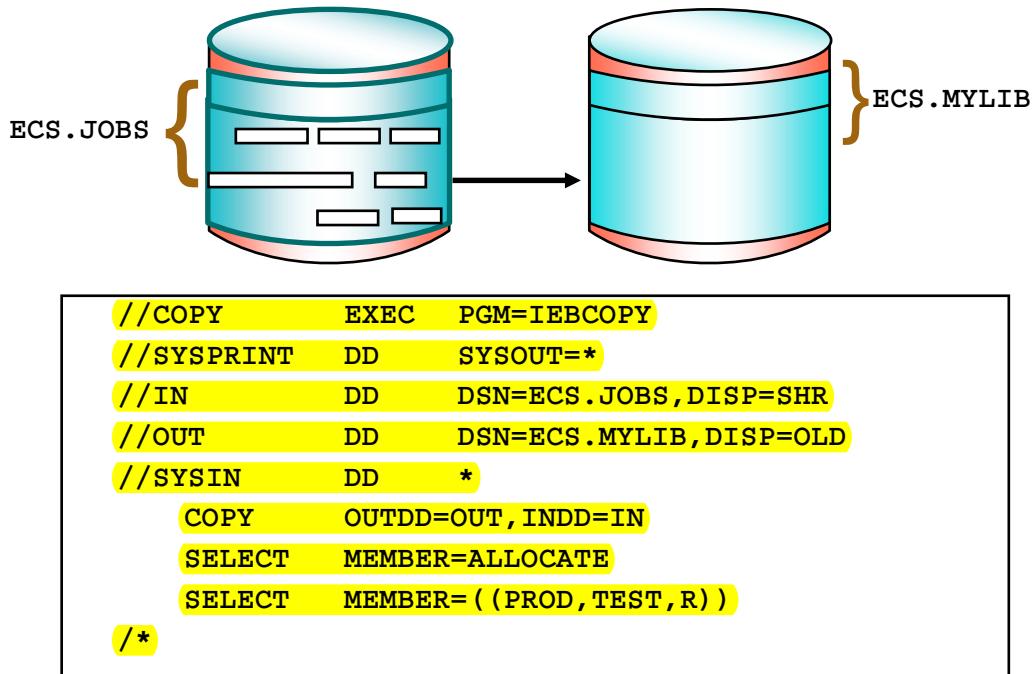
Notes:

The step outlined in the visual executes IEBGENER to copy the sequential input data set to a new partitioned output data set.

- The EXEC statement specifies the program to be executed.
- The SYSPRINT DD statement defines the message data set.
- The SYSUT1 DD statement defines the input data set.
- The SYSUT2 DD statement defines the output data set.
- The SYSIN DD statement defines the control data set. This is where IEBGENER looks for utility control statements. If none are required, as in this example, //SYSIN DD DUMMY can be coded.

For further information about IEBGENER, refer to *DFSMSdfp Utilities*, SC26-7414.

IEBCOPY copy



© Copyright IBM Corporation 2011

Figure 8-3. IEBCOPY copy

ES074.0

Notes:

IEBCOPY is a data set utility used to copy or merge members between one or more partitioned data sets (PDSs), or partitioned data sets extended (PDSEs), in full or in part. You can also use IEBCOPY to create a backup of a PDS into a sequential data set (called an *unload data set* or *PDSU*), and to copy members from the backup into a PDS.

The step outlined in the visual executes IEBCOPY to copy two members of the PDS ECS.JOBS to an existing PDS, ECS.MYLIB.

- The EXEC statement specifies program to be executed.
- The SYSPRINT DD statement defines the message data set.
- The IN and OUT DD statements define data sets to be used by IEBCOPY.
- The SYSIN DD * statement indicates that instream records, or control statements, follow.
- The COPY control statement specifies the input and output DDNAMES.

- The first SELECT control statement specifies the member ALLOCATE is to be copied from the input to the output data set.
- The format of the first SELECT control statement is:
`SELECT MEMBER=NAME OR SELECT MEMBER=(NAME, NAME, NAME)`
- The second SELECT control statement specifies:
 - The member PROD is to be copied from the input data set.
 - Rename PROD to TEST.
 - Copy the renamed member TEST to the output data set.
 - If a member name TEST currently exists in the output data set, replace it.
- The format of the second SELECT control statement is:
`SELECT MEMBER= ((NAME, NEWNAME, REPLACE))`

IEBCOPY listing

The diagram shows a terminal window displaying an IEBCOPY listing. A callout bubble on the left points to the control statements with the text "Copy two members from the input PDS to output PDS". Another callout bubble on the right points to the statistics with the text "Unused tracks and directory blocks".

```

IEBCOPY messages and control statements
COPY OUTDD=OUT, INDD=IN
SELECT MEMBER=ALLOCATE
SELECT MEMBER=((PROD,TEST,R))

IE81671 FOLLOWING MEMBERS COPIED FROM INPUT DATA SET REFERENCED BY IN
IEB1541 ALLOCATE HAS BEEN SUCCESSFULLY COPIED
IEB1441 THERE ARE 0000013 UNUSED TRACKS IN OUTPUT DATA SET REFERENCED BY OUT
IEB1491 THERE ARE 0000042 UNUSED DIRECTORY BLOCKS IN OUTPUT DIRECTORY
IEB1471 END OF JOB -00 WAS HIGHEST SEVERITY CODE
IEB1551 TEST HAS BEEN SUCCESSFULLY COPIED AND IS A NEW NAME

```

© Copyright IBM Corporation 2011

Figure 8-4. IEBCOPY listing

ES074.0

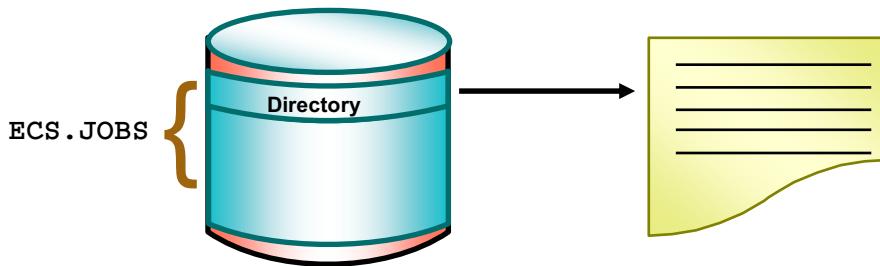
Notes:

IEBCOPY prints the control statements then specifies which members were copied from the input to the output data set.

In this case, both members were copied to ECS.MYLIB.

IEBCOPY also tells how many unused tracks and directory blocks are in the output PDS after the copy operation.

IEHLIST LISTPDSE



```
//LISTPDS EXEC PGM=IEHLIST
//SYSPRINT DD SYSOUT=*
//NUM1 DD UNIT=SYSDA,VOL=SER=WORK01,DISP=OLD
//SYSIN DD *
      LISTPDS DSNAME=ECS.JOBS,VOL=SYSDA=WORK01
/*
```

© Copyright IBM Corporation 2011

Figure 8-5. IEHLIST LISTPDSE

ES074.0

Notes:

The **IEHLIST** utility is used to list a PDS directory or a disk volume VTOC. **IEHLIST** can be used to list entries in a PDS directory.

- The **JOB** statement introduces the job/processing requirements.
- The **EXEC** statement specifies the program to be executed.
- The **SYSPRINT DD** statement defines the message data set.
- The **NUM1 DD** statement allocates the pack **WORK01** needed for the **LISTPDS** operation. (**NUM1** can be any legal name you choose.)
- **SYSIN DD *** statement defines the control data set. This is where **IEHLIST** looks for utility control statements.
- The **LISTPDS** control statement requests a listing of the directory for the PDS **ECS.JOBS**.



Note

DSNAME *cannot* be abbreviated as DSN on a control statement.

The VOL parameter format is:

VOL=XXXXX=YYYYYYY

Where XXXXX is the value specified for the UNIT parameter on the JCL DD statement and YYYYYYY is the SER subparameter value.

LISTPDS listing

The list of PDS members and their addresses

SYSTEMS SUPPORT UTILITIES --- IEHLIST							PAGE 1
DATE: 1988.168 TIME: 10.11.01 DIRECTORY INFO FOR SPECIFIED PDS ON VOL WORK01							
ECS.JOBS							
MEMBERS TTRC VARIABLE USERID DATA --- (USER DATA AND TTRC ARE IN HEX)							
ALLOCATE 0001020F	0102000000	88105F0088	106F082500	0C00060000	E3E2D6E2E2	D9C2404040	
COMPRESS 00002D00							
DELETE 0001040F	0103000000	88105F0088	106F083000	0600060000	E3E2D6E2E2	D9C2404040	
GENER1 00010C0F	0105000000	88106F0088	141F200300	0B00070008	E3E2D6E232	D9C2404040	
GEMER2 00010EOF	0103000000	88106F0088	141F201800	0D0007000A	E3E2D6E232	D9C2404040	
GENER3 00010A0F	0102000000	88133F0088	141F152300	0D000C0007	E3E2D6E232	D9C2404040	
IEBPTPCH 0001100F	010F000000	88140F0088	144F190600	1200110001	E3E2D6E232	D9C2404040	
IEFBR14A 0001120F	0102000000	88105F0088	162F163600	0300060000	E3E2D6E232	D9C2404040	
IEFBR14D 00002F0F	0103000000	88105F0088	105F161100	0400060000	E3E2D6E232	D9C2404040	
LISTCAT 0001060F	0107000000	88105F0088	112F194300	0800090002	E3E2D6E232	D9C2404040	
LISTPDS 0001140F	010A000000	88105F0088	168F100700	0900070000	E3E2D6E232	D9C2404040	
LISTVTOC 0001080F	0105000000	88105F0088	141F143800	0600060006	E3E2D6E232	D9C2404040	

OF THE 00043 DIRECTORY BLOCKS ALLOCATED TO THIS PDS, 00040 ARE (IS) COMPLETELY UNUSED

Allocated and unused directory blocks

© Copyright IBM Corporation 2011

Figure 8-6. LISTPDS listing

ES074.0

Notes:

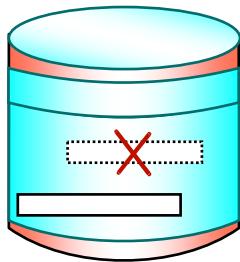
This is a list of all members with the PDS named ECS.JOBS.

Member names are in alphabetical order.

Actual track address of the members are given under the TTRC.

This list specifies the number of allocated and unused directory blocks in the PDS.

IEHPROGM SCRATCH/RENAME



```
//TSOMJ00      JOB     MSGCLASS=A
//PDSSTUFF    EXEC    PGM=IEHPROGM
//SYSPRINT    DD      SYSOUT=*
//NUM1        DD      UNIT=SYSDA,VOL=SER=WORK01,DISP=OLD
//SYSIN        DD      *
      SCRATCH MEMBER=ALLOCATE,DSNAME=ECS.MYLIB,           X
                  VOL=SYSDA=WORK01
      RENAME   MEMBER=COMPRESS,DSNAME=ECS.MYLIB,           X
                  VOL=SYSDA=WORK01,NEWNAME=COPYPDS
/*

```

© Copyright IBM Corporation 2011

Figure 8-7. IEHPROGM SCRATCH/RENAME

ES074.0

Notes:

Most of the **IEHPROGM** functions are available in **IDCAMS**, and that is now the preferred utility for catalog and data set functions.

IEHPROGM has many uses. It can be used to scratch (delete) a data set, or it can be used to scratch or rename a member of a PDS. **IEHPROGM** will also catalog and uncatalog data sets.

In this example, **IEHPROGM** is used to scratch the member **ALLOCATE** in the PDS **ECS .MYLIB**. The **RENAME** control statement asks **IEHPROGM** to rename **COMPRESS** to **COPYPDS**.

IEHPROGM listing

SCRATCH ALLOCATE

```
SYSTEM SUPPORT UTILITIES ---- IEHPROGM
SCRATCH MEMBER=ALLOCATE,DSNAME=ECS.MYLIB,
          VOL=SYSDA=WORK01
NORMAL END OF TASK RETURNED FROM SCRATCH
```

X

```
→ RENAME MEMBER=COMPRESS,DSNAME=ECS.MYLIB,
          VOL=SYSDA=WORK01,NEWNAME=COPYPDS
NORMAL END OF TASK RETURNED FROM RENAME
UTILITY END
```

X

RENAME COMPRESS

© Copyright IBM Corporation 2011

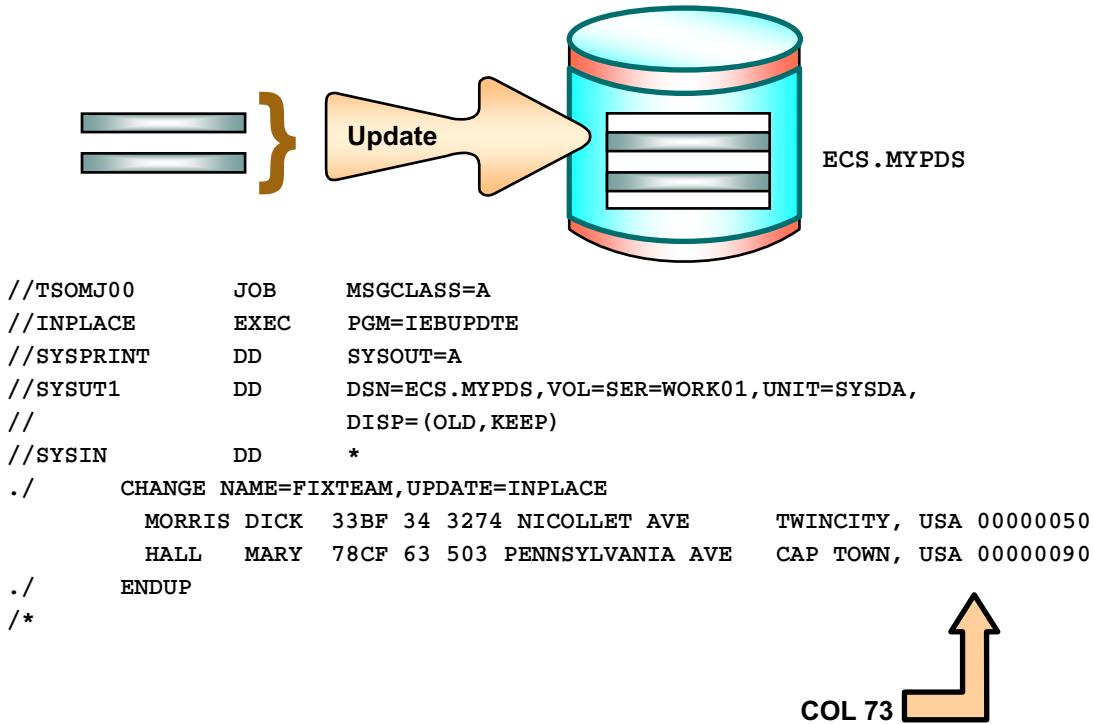
Figure 8-8. IEHPROGM listing

ES074.0

Notes:

IEHPROGM prints the control statements it reads; it then specifies whether it successfully accomplished the operation. In this case, IEHPROGM successfully did the scratch and rename.

IEBUPDTE update in place



© Copyright IBM Corporation 2011

Figure 8-9. IEBUPDTE update in place

ES074.0

Notes:

IEBUPDTE is used to create, update, and modify sequential data sets and members of partitioned data sets. The sequential data sets or members have logical record lengths not exceeding 80 bytes.

The IEBUPDTE utility uses control statements within the first two columns.

When a PDS member is changed and then replaced or added to a PDS, it normally goes in the available space after (following) the last member in the PDS. If several records in a member are replaced by an equal number of records, IEBUPDTE can update the member without changing its address in the PDS. This is called *update in place*.

In the example in the visual, the PDS ECS .MYPDS contains a member named FIXTEAM. FIXTEAM contains two records with sequence numbers (00000050 and 00000090) in columns 73 to 80. When the IEBUPDTE job is executed, those two records will be replaced by the two records in the SYSIN input stream. This is an update in place.

IEBUPDTE print output

Before run

Replace →

JONES FRED 53AF	87 5701 NINE MILE ROAD	AUTOCITY, USA	00000010
ANDERSONDON 78AF	34 320 WESTHEIMER, #219	OIL CITY, USA	00000020
WILSON SAM 53AF	87 4745 HARRY HINES BLVD	TV TOWN, USA	00000030
WILLIAMSANN 69CF	63 5622 PARK AVENUE, #368	BIG CITY, USA	00000040
HARRIS CLARA 78AF	87 324 MICHIGAN AVENUE	LAKECITY, USA	00000050
MATHEWS SUE 78AF	34 6700 WILSHIRE BLVD	STARCITY, USA	00000060
SMITH BILL 53BF	87 #35 LOMBARD	HILLCITY, USA	00000070
LEWIS ELLIE 33CF	63 5346 PEACH TREE STREET	SOUTH TOWN, USA	00000080
SIMS CINDY 69BF	34 5478 MARKET STREET	BELL TOWN, USA	00000090
CHARLES BOB 69CF	63 2300 BAY STREET	OLD CITY, USA	00000100

Replace →

IEBUPDTE run

SYSIN	NEW MASTER	IEBUPDTE LOG PAGE 0001
./ CHANGE NAME=FIXTEAM,UPDATE=INPLACE		
HARRIS CLARA 78AF	87 324 MICHIGAN AVENUE	LAKECITY, USA 00000050 REPLACED
MORRIS DICK 33BF	34 3274 NICOLLET AVE,	TWIN CITY, USA 00000050REPLACEMENT*
SIMS CINDY 69BF	34 5478 MARKET STREET	BELL TOWN, USA 00000090 REPLACED
HALL MARY 78CF	63 503 PENNSYLVANIA AVE	CAP TOWN, USA 00000090REPLACEMENT*
./ ENDUP		

IEB818 HIGHEST CONDITION CODE WAS 00000000

IEB8191 END OF JOB IEBUPDTE.

After run

Replaced →

JONES FRED 53AF	87 5701 NINE MILE ROAD	AUTOCITY, USA	00000010
ANDERSONDON 78AF	34 320 WESTHEIMER, #219	OIL CITY, USA	00000020
WILSON SAM 53AF	87 4745 HARRY HINES BLVD	TV TOWN, USA	00000030
WILLIAMSANN 69CF	63 5622 PARK AVENUE, #368	BIG CITY, USA	00000040
MORRIS DICK 33BF	34 3274 NICOLLET AVE,	TWIN CITY, USA	00000050
MATHEWS SUE 78AF	34 6700 WILSHIRE BLVD	STARCITY, USA	00000060
SMITH BILL 53BF	87 #35 LOMBARD	HILLCITY, USA	00000070
LEWIS ELLIE 33CF	63 5346 PEACH TREE STREET	SOUTH TOWN, USA	00000080
HALL MARY 78CF	63 503 PENNSYLVANIA AVE	CAP TOWN, USA	00000090
CHARLES BOB 69CF	63 2300 BAY STREET	OLD CITY, USA	00000100

© Copyright IBM Corporation 2011

Figure 8-10. IEBUPDTE print output

ES074.0

Notes:

The top diagram in the visual shows the contents of FIXTEAM before executing the IEBUPDTE job. Note the sequence numbers in columns 73 to 80.

The middle diagram shows the printed output from the IEBUPDTE run. It shows both of the original records that are to be replaced as well as their replacements.

The bottom diagram shows the contents of FIXTEAM after executing the IEBUPDTE job.

The BPXBATCH utility

Shell script in batch

```
//SHELLJOB JOB (MARIANNE),MSGCLASS=A,MSGLEVEL=(1,1)
//SHSTEP      EXEC PGM=BPXBATCH
//STDIN        DD PATH='/u smith/bin/myscript.in',PATHOPTS=(ORDONLY)
//STDOUT       DD PATH='/u smith/bin/mystd.out',PATHOPTS=(OWRONLY,OCREATE),
//                           PATHMODE=SIRWXU
//STDERR       DD PATH='/u smith/bin/mystd.err',PATHOPTS=(OWRONLY,OCREATE),
//                           PATHMODE=SIRWXU
//STDENV       DD *
path=/bin:/u/usr/joeuser
/*
```

**MVS
Started
Task (2)**

```
//INETD  PROC
//INETD  EXEC PGM=BPXBATCH,REGION=30M,TIME=NOLIMIT,
//          PARM='PGM /usr/sbin/inetd /etc/inetd.conf'
//** STDIN and STDOUT are both defaulted to /dev/null
//STDERR  DD PATH='/etc/log',
//          PATHOPTS=(OWRONLY,OCREATE,OAPPEND),
//          PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//STDENV  DD PATH='/etc/std.env',PATHOPTS=(ORDONLY)
```

© Copyright IBM Corporation 2011

Figure 8-11. The BPXBATCH utility

ES074.0

Notes:

The BPXBATCH utility can be used to run shell scripts, or C programs resident in the HFS, as a batch job in a JES initiator. After the BPXBATCH job step program has set up the support environment, it issues `exec()` to replace BPXBATCH with the program or shell script to be run.

The BPXBATCH utility provides support to run shell scripts or z/OS UNIX application programs as MVS batch jobs. BPXBATCH can be invoked in JCL as an MVS batch job in addition to other ways (TSO/E command, in REXX, with OSHELL).

z/OS UNIX C programs require that `stdin`, `stdout`, and `stderr` be defined. Many C functions also use this to define input and output.

For BPXBATCH, the default for `stdin` and `stdout` is `/dev/null`, and `stderr` is the same as `stdout`. This can be changed to file names by using the DD names `STDIN`, `STDOUT`, and `STDERR`.

The second example in the visual shows a typical way to start a UNIX daemon using the BPXBATCH program to invoke the daemon program.

Checkpoint (1 of 2)

1. Which of the following are valid statements for input data set with IEBCOPY?
 - a. //IN DD DSN=...
 - b. //SYSIN DD DSN=...
 - c. //SYSUT1 DD DSN=...
 - d. All of them

2. With the IEBCOPY input statement, what is the meaning of the R in the following example: SELECT MEMBER= ((NAME1, NAME2, R))

3. IEBCOPY can be used to:
 - a. Compress a PDS
 - b. Rename members of a PDS
 - c. Replace members of a PDS
 - d. Delete members of a PDS
 - e. All of the above

© Copyright IBM Corporation 2011

Figure 8-12. Checkpoint (1 of 2)

ES074.0

Notes:

Checkpoint (2 of 2)

4. True or False: IEHPROGM is the recommended utility to delete/scratch data sets.
5. What utility can be used to update and modify sequential data sets and members of partitioned data sets?
6. What is the name of the input DD statement when you execute UNIX shell scripts in batch?

© Copyright IBM Corporation 2011

Figure 8-13. Checkpoint (2 of 2)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Look at examples of more utility programs:
 - IEBGENER
 - IEBCOPY
 - IEHLIST
 - IEHPROGM
 - IEBUPDTE
 - BPXBATCH
- Use the *DFSMS/dfp Utilities* manual for reference

© Copyright IBM Corporation 2011

Figure 8-14. Unit summary

ES074.0

Notes:

Unit 9. More on procedures

What this unit is about

This topic introduces procedure enhancements added in previous releases of z/OS.

What you should be able to do

After completing this unit, you should be able to:

- Compare procedures and INCLUDE groups
- Describe the purpose of the JCLLIB statement
- Discuss the INCLUDE statement and INCLUDE groups
- Describe the benefits of nested procedures
- Describe the use of the SET statement
- Compare the PROC, EXEC, and SET statements for supplying symbolic parameters

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597	<i>z/OS MVS JCL Reference</i>
SA22-7598	<i>z/OS MVS JCL User's Guide</i>

Unit objectives

After completing this unit, you should be able to:

- Compare procedures and INCLUDE groups
- Describe the purpose of the JCILLIB statement
- Discuss the INCLUDE statement and INCLUDE groups
- Describe the benefits of nested procedures
- Describe the use of the SET statement
- Compare the PROC, EXEC, and SET statements for supplying symbolic parameters

© Copyright IBM Corporation 2011

Figure 9-1. Unit objectives

ES074.0

Notes:

Procedures and INCLUDE groups

- Procedures
 - Consist of multiple executable steps
 - Accessed by the EXEC statement

- INCLUDE groups
 - Consist of a sequence of JCL statements
 - Accessed by INCLUDE statement

Note: Before z/OS V1R13 neither procedures nor INCLUDE groups could contain instream data. From z/OS V1R13 and JES2 V1R13, instream data is supported.

© Copyright IBM Corporation 2011

Figure 9-2. Procedures and INCLUDE groups

ES074.0

Notes:

For jobs that you run frequently or types of jobs that use the same job control, you can prepare sets of job control statements, called procedures.

- Procedures consist of one or more complete steps.
- Every procedure must be given a name.
- Procedures can be invoked via the EXEC statement.
- Procedures are supported in every version and release z/OS.
- Procedures can be nested up to 15 levels

INCLUDE groups are similar to a procedure in that both consist of JCL statements; however, an INCLUDE group need not consist of complete steps.

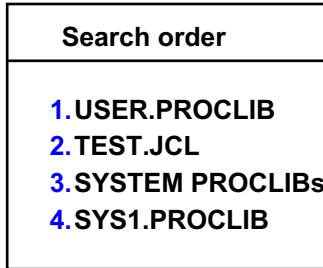
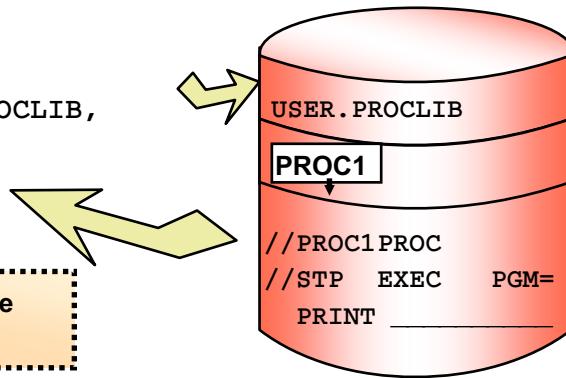
- An INCLUDE group consists of a sequence (list) of JCL statements.
- An INCLUDE group need not consist of complete steps.
- INCLUDE groups are accessed through the INCLUDE statement.

- When accessed, the INCLUDE statements are inserted into the job stream in the order they appear in the INCLUDE group.
- INCLUDE groups are supported in z/OS.
- Three benefits of using INCLUDE groups are they:
 - Save time by reducing the time required to code JCL.
 - Save library storage by eliminating duplicate JCL.
 - Reduce JCL errors by providing access to debugged JCL.

Accessing the PROC: JCLLIB statement

```
//JOB1 JOB      MSGCLASS=A
//UPLIB JCLLIB  ORDER=(USER.PROCLIB,
//                      TEST.JCL)
//STP1 EXEC     PROC=PROC1
```

PROC1 JCL statements are expanded here



© Copyright IBM Corporation 2011

Figure 9-3. Accessing the PROC: JCLLIB statement

ES074.0

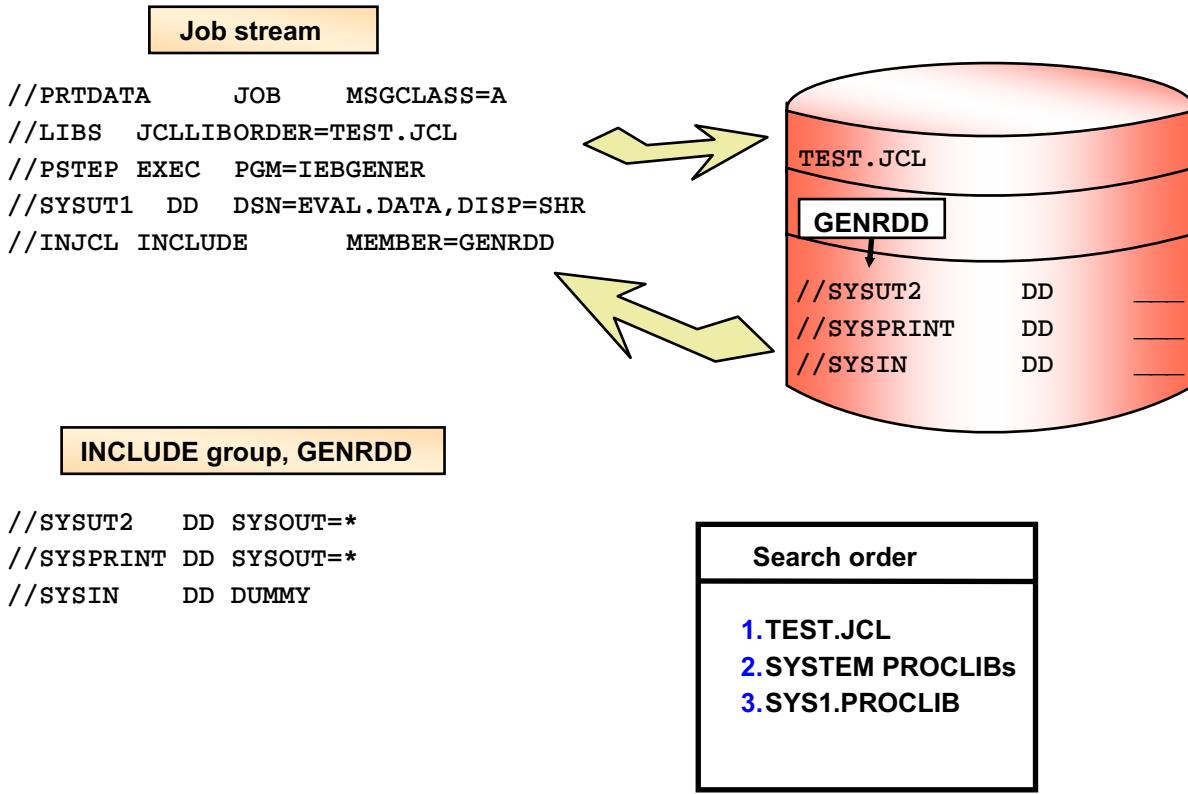
Notes:

Note the following principles regarding JCLLIB statements.

- The JCLLIB statement identifies the names of private libraries the system searches to locate:
 - Procedures invoked in the job.
 - INCLUDE groups invoked in the job.
- The format for the JCLLIB statement is:


```
//UPLIB    JCLLIB   ORDER=library
//UPLIB    JCLLIB   ORDER=(library,.....)
```
- The system searches these libraries (or proclibs) in the following order:
 - Libraries listed on the JCLLIB statement (in their order of listing)
 - Installation-defined proclibs
 - SYS1.PROCLIB
- Only one JCLLIB statement can be coded per job. It must appear after the JOB statement and before the first EXEC statement.

The INCLUDE statement (1 of 2)



© Copyright IBM Corporation 2011

Figure 9-4. The INCLUDE statement (1 of 2)

ES074.0

Notes:

An *INCLUDE group* is a sequence of JCL statements (such as DD and OUTPUT JCL statements) that do not necessarily comprise complete steps.

The INCLUDE group is accessed in JCL through the INCLUDE statement.

The INCLUDE group replaces the INCLUDE statement, and the system processes the imbedded JCL statements as part of the JCL stream.

The INCLUDE statement can be coded anywhere after the JOB statement. The format for the INCLUDE statement is:

```
//INJCL INCLUDE MEMBER=NAME
```

Only one member name can be coded, the name of the INCLUDE group. The location of the INCLUDE group can be specified on the JCLLIB statement, or the INCLUDE group can be placed in an installation-defined proclib or SYS1.PROCLIB.

An INCLUDE statement can appear inside an INCLUDE group. This allows INCLUDE groups to be nested to a maximum of 15 levels.

INCLUDE groups cannot contain the following JCL statements:

- JOB statements
- PROC or PEND statements
- JCLLIB statements
- DD * or DD DATA statements
- JES2 or JES3 control statements or JES commands

include within a proc, but not proc within include.

The INCLUDE statement (2 of 2)

Resulting JCL

```
//PRTDATA   JOB      MSGCLASS=A  
  
//LIBS       JCLLIB   ORDER=TEST.JCL  
  
//PSTEP      EXEC     PGM=IEBGENER  
  
//SYSUT1     DD       DSN=EVAL.DATA,DISP=SHR  
  
//SYSUT2     DD       SYSOUT=*  
  
//SYSPRINT   DD       SYSOUT=*  
  
//SYSIN      DD       DUMMY
```

© Copyright IBM Corporation 2011

Figure 9-5. The INCLUDE statement (2 of 2)

ES074.0

Notes:

This is the resulting JCL from the previous visual.

Note that the system located the INCLUDE group in the library named on the JCLLIB statement.

The three JCL statements in the INCLUDE group replace the INCLUDE statement in the original job.

Nested example

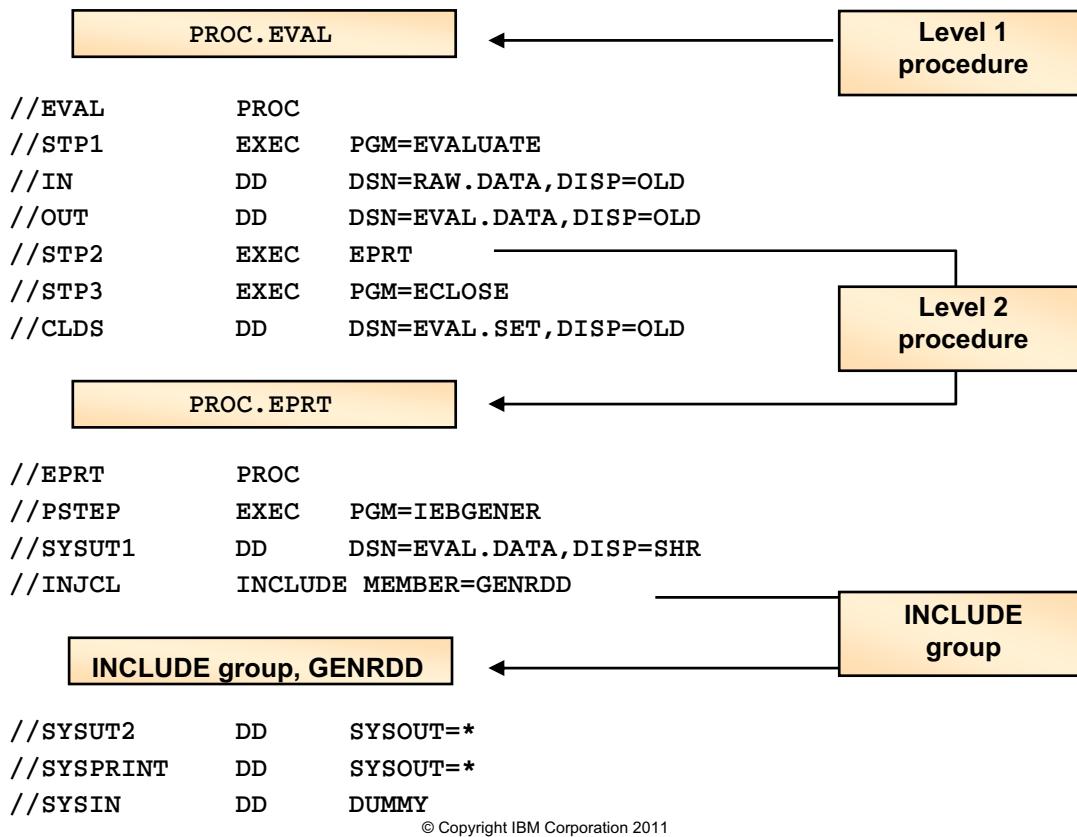


Figure 9-6. Nested example

ES074.0

Notes:

Procedures can be nested to a maximum of 15 levels.

INCLUDE groups can also be nested to a maximum of 15 levels.

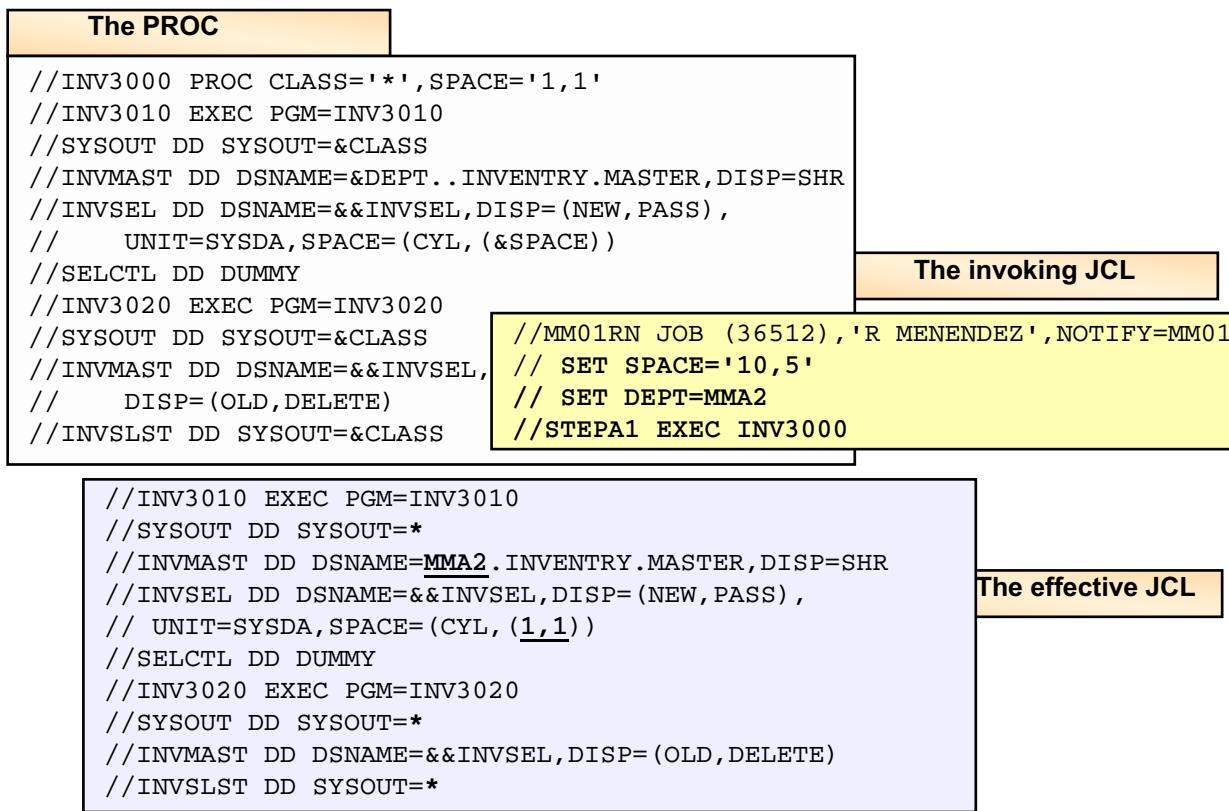
A procedure can access an INCLUDE group.

An INCLUDE group can access a procedure.

This example shows the procedure EVAL, which has a procedure named EPRT nested within it.

In addition, the procedure EPRT has an INCLUDE group GENRDD nested within it.

SET statement



© Copyright IBM Corporation 2011

Figure 9-7. SET statement

ES074.0

Notes:

The SET statement assigns values to symbolic parameters in JCL. The symbolic parameters can appear in procedures or in general JCL.

The SET statement can appear anywhere in JCL between the JOB statement and the first point where a SET statement-assigned symbolic parameter is referenced.

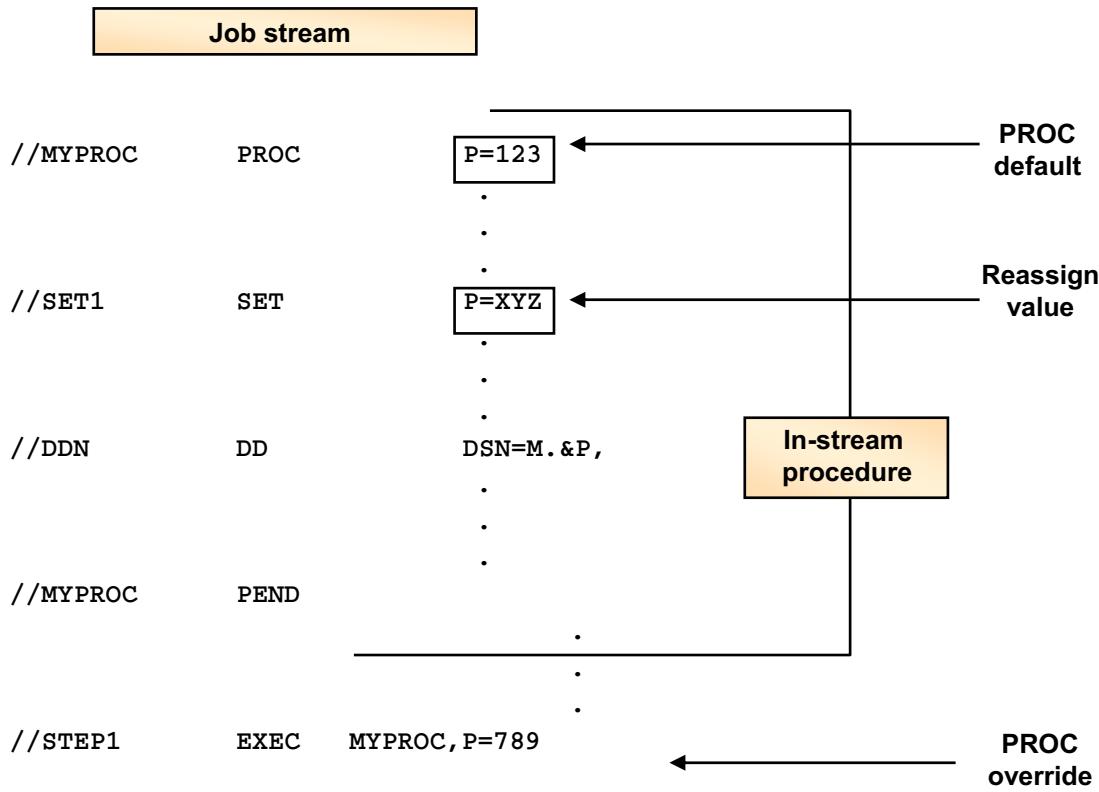
Multiple SET statements can assign a value to a symbolic parameter. In this case, the value assigned on the last SET statement is used.

The SET statement is another way to assign values to symbolic parameters.

In contrast to a PROC statement that assigns default values or a procedure-invoking EXEC statement that provides runtime values, the SET statement lets you set the values of symbolic parameters at any point or time within your JCL.

Be aware, however, that SET values are overridden by any values that are assigned in subsequent PROC or EXEC statements in the job.

Overriding symbolic parameters



© Copyright IBM Corporation 2011

Figure 9-8. Overriding symbolic parameters

ES074.0

Notes:

- Symbolic parameters can be assigned through three statements:
 - PROC statement:** Supplies procedure defaults.
 - EXEC statement:** Overrides procedure defaults.
 - SET statement:** Supplies values in procedure and general JCL.
- PROC statement-assigned symbolic parameter values are always overridden by the values assigned on the calling EXEC statement.
- A value you assign to a symbolic parameter on a SET statement is overridden by:
 - A subsequent SET statement.
 - Any default value assigned or nullified on a subsequent PROC statement for a procedure.
 - Any value assigned or nullified on a subsequent EXEC statement that calls a procedure.
- In the example in the visual, the symbolic parameter &P has the value of 789 assigned on the invoking STEP1 EXEC statement.
- The SET1 SET statement reassigns the value of &P to XYZ.
- Therefore, on the DDN DD statement, the data set name is M.XYZ.

Symbols and JCL

Static System Symbols	Dynamic System Symbols	Symbols Reserved For System Use
&SYSCLOSE &SYSNAME &SYSPLEX &SYSR1 &SYSLV	&DATE &DAY &HHMMSS &HR &JDAY &JOBNAME &MIN &MON &SEC &SEQ &TIME &WDAY &YR2 &YR4 &YYMMDD	&DATE &HR &LYYMMDD &LHR &LMON &LWDAY &LHHMMSS &SEC &SYSCLOSE &SYSR1 &WDAY &YYMMDD
Installation Defined System Symbols		
JCL Symbol		
IPCS Symbol		

```
//SMTP PROC MODULE=SMTP,DEBUG=,PARMS=' / ',SYSERR=SYSERR
//SETMSG EXEC PGM=SETMSG,PARM=ON
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SMTP EXEC PGM=MVPMAIN,REGION=6144K,TIME=1440,
// *PARM='&MODULE,PARM=&DEBUG,ERRFILE(&SYSERR),&PARMS'
//STEPLIB DD DSN=SYS1.SEZATCP,DISP=SHR
//*SMTPNJE DD DSN=TCPIP.SMTPNJE.HOSTINFO,DISP=SHR
//CONFIG DD DSN=&SYSNAME..TCPIP.PARMLIB(SMTP&SYSNAME.),DISP=SHR
```

© Copyright IBM Corporation 2011

Figure 9-9. Symbols and JCL

ES074.0

Notes:

The system allows you to define and use the following types of symbols:

- **JCL symbol:** A symbol that represents variable information in JCL. You can define JCL symbols on EXEC, PROC, and SET statements in JCL, and use them only in:
 - JCL statements in the job stream
 - Statements in cataloged or instream procedures
 - DD statements that are added to a procedure
- **Dynamic system symbol:** A system symbol whose substitution text can change at any point in an IPL. Dynamic system symbols represent values that can change often, such as dates and times. A set of dynamic system symbols is defined to the system; your installation cannot provide additional dynamic system symbols.
- **Static system symbol:** A symbol whose substitution text is defined at system initialization and remains fixed for the life of an IPL. Static system symbols are used to represent fixed values such as system names and sysplex names.

Static system symbols have two types:

- *System-defined static system symbols* already have their names defined to the system. Your installation defines substitution texts or accepts system default texts for the static system symbols, which are:
 - &SYSCLONE
 - &SYSNAME
 - &SYSPLEX
 - &SYSR1
 - &SYSALVL
- *Installation-defined static system symbols* are defined by your installation. The system programmer specifies their names and substitution texts in the SYS1.PARMLIB data set (IEASYMxx member).

The figure in the visual shows us the mix of using both JCL symbols (in red) and system symbols (in blue). This sample represents a JCL PROC to start an SMTP server.



Warning

Note that you can use static system symbols such as &SYSNAME in started task JCL (PROC) but not in regular batch JCL.

Checkpoint (1 of 2)

1. What is the JCL statement required to support include groups?
 - a. JCLIN
 - b. INJCL
 - c. JCLINPUT
 - d. JCLLIB

2. True or False: An INCLUDE group must contain at least one EXEC statement.

3. How many member names can be coded in an INCLUDE statement?
 - a. One
 - b. Eight
 - c. No limit

4. True or False: An INCLUDE group can contain a PROC.

5. True or False: A PROC can contain an INCLUDE statement.

© Copyright IBM Corporation 2011

Figure 9-10. Checkpoint (1 of 2)

ES074.0

Notes:

Checkpoint (2 of 2)

6. True or False: A SET statement must appear right after the JOB statement.
7. True or False: You can use static symbols like &SYSNAME in batch JCL.
8. Multiple SET statements can assign a value to a symbolic parameter. In this case, the value assigned on which SET statement is used?
 - a. First
 - b. Last
9. Which of the following is a valid system static symbol?
 - a. &SYSUID
 - b. &SYSCLONE
 - c. &JOBNAME

© Copyright IBM Corporation 2011

Figure 9-11. Checkpoint (2 of 2)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Compare procedures and INCLUDE groups
- Describe the purpose of the JCILLIB statement
- Discuss the INCLUDE statement and INCLUDE groups
- Describe the benefits of nested procedures
- Describe the use of the SET statement
- Compare the PROC, EXEC, and SET statements for supplying symbolic parameters

© Copyright IBM Corporation 2011

Figure 9-12. Unit summary

ES074.0

Notes:

Unit 10. Selected JCL topics

What this unit is about

Advanced topics, such as the IF/THEN/ELSE/ENDIF statement construct, OUTPUT processing, SYSOUT distribution, and additional parameters, are covered in this unit.

What you should be able to do

After completing this unit, you should be able to:

- Describe conditional execution of job steps using the IF/THEN/ELSE/ENDIF statement construct
- Code the parameter to control the amount of output to be printed and the timing of the printing
- Use the OUTPUT statement to route SYSOUT to multiple nodes and to establish default forms control for SYSOUT data sets
- Discuss the SYSOUT distribution parameters
- Discuss the disposition of a SYSOUT data set
- Describe the NOTIFY parameter

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597

z/OS MVS JCL Reference

SA22-7598

z/OS MVS JCL User's Guide

Unit objectives

After completing this unit, you should be able to:

- Describe conditional execution of job steps using the IF/THEN/ELSE/ENDIF statement construct
- Code the parameter to control the amount of output to be printed and the timing of the printing
- Use the OUTPUT statement to route SYSOUT to multiple nodes and to establish default forms control for SYSOUT data sets
- Discuss the SYSOUT distribution parameters
- Discuss the disposition of a SYSOUT data set
- Describe the NOTIFY parameter

© Copyright IBM Corporation 2011

Figure 10-1. Unit objectives

ES074.0

Notes:

Conditional execution of JOB steps

- **Technique 1:** The COND parameter
 - Specifies when a step should *not* execute
 - Coded on the JOB or EXEC statement

- **Technique 2:** The IF/THEN/ELSE/ENDIF statement
 - Specifies when a step should execute
 - Placed anywhere after the JOB statement

© Copyright IBM Corporation 2011

Figure 10-2. Conditional execution of JOB steps

ES074.0

Notes:

z/OS provides facilities to conditionally execute job steps depending on the outcome of previous job steps. Two techniques are provided for testing the success or failure of job steps and determining whether to continue execution based on the outcome of the tests:

- COND parameter
- IF/THEN/ELSE/ENDIF statement construct

Using the IF construct is more powerful than COND.

- The COND parameter on the first step of a job is ignored.
 - However, the IF construct is tested.
- We can code symbolic parameters in the IF conditions
- We can code many types of relational expressions in IF condition

The IF/THEN/ELSE/ENDIF statement construct introduces relational expressions into z/OS JCL. Use these expressions to specify tests that prior steps must satisfy. This provides an alternative to the COND parameter that is easier to understand.

You can nest IF/THEN/ELSE/ENDIF statement constructs up to a maximum of 15 levels.

Relational expressions

- Relational expressions are constructed from:
 - Not operator
 - NOT
 - Comparison operators
 - GT, LT, NG, NL, EQ, NE, GE, LE
 - Logical operators
 - AND, OR
 - Relational expression keywords
 - RC, ABEND, RUN, and so on

© Copyright IBM Corporation 2011

Figure 10-3. Relational expressions

ES074.0

Notes:

Note the following principles regarding relational expressions:

- The condition (relational expression) consists of:
 - Comparison operators
 - Logical operators
 - Not (\neg) operators
 - Relational expression keywords.
- The relational expression is constructed from relational expression keywords and the three types of operators listed shown above.
- Operators from one or more of the three classes above can appear in a single relational expression. If several operators appear in a relational express, they are evaluated in the order:
 - **NOT operator:** Evaluated first
 - **Comparison operators:** Evaluated second
 - **Logical operators:** Evaluated third

- By using relational expression keywords a relational expression can:
 - Compare a return code against a coded value by using the comparison operators
 - Determine whether a given prior step did or did not ABEND
 - Check for a specific system ABEND completion code or user-defined ABEND completion code
 - Determine whether a given prior step did or did not execute
- Several simple relational expressions can be combined using logical operators. For example, the expression 'RC > 4' can be combined with the expression 'RC < 8' to produce the compound expression 'RC > 4 & RC < 8'.
- Two different formats can be used for each operator in a relational expression. For example, greater than can be represented by 'GT' or '>', and AND can be represented by 'AND' or '&'. If the alphabetic form (such as GT or AND) is used, a blank must precede and follow the operator in a relational expression. If the non-alphabetic format (for example, > or &) is used, the blanks are optional.

The following keywords are the only keywords supported by IBM and recommended for use in relational expressions. Any other keywords, even if accepted by the system, are not intended or supported keywords.

- RC indicates a return code.
- ABEND indicates that an abend condition occurred.
- \neg ABEND indicates that no abend condition occurred.
- ABENDCC indicates a system or user completion code.
- RUN indicates that the specified step started execution.
- \neg RUN indicates that the specified step did not start execution.

Keywords used for condition-checking are ABEND and RUN.

- Ex: // IF (step1.ABEND)
- Or // IF (step1.RUN)

This is same as // IF (step1.ABEND=TRUE) and // IF (step1.RUN=TRUE).

We can also check the reverse condition as // IF (step1.ABEND=FALSE) and // IF (step1.RUN=FALSE).

We can also check for system and user completion codes:

- // IF (step1.ABENDCC=U4039) or // IF (step1.ABENDCC=SB37)

All the keywords can include a step name and proc step name to refine the test to a specific job step.

- The format is *stepname.procstepname.keyword*.

RC checks a return code.

Example JCL:

```
IF  RC = 0    THEN
IF  STEP1.RC < 12  THEN
```

If you have not given the step name, the highest return code from all job steps is taken for checking.

ABENDCC checks a user/system ABEND condition code

Example JCL:

```
IF ABENDCC = S0C7 THEN
```

Suppose you want to check error code of particular step, given step name.ABENDCC. If you have not given the step name, the most recent ABEND code that occurred is taken for checking.

ABEND checks for an abnormal end of a program

Example JCL:

```
IF ABEND THEN  
IF STEP4.PROCAS01.ABEND = TRUE THEN
```

If you have not given any step name, all steps prior to this condition will be checked.

RUN checks whether a job step executed or not

Example JCL:

If you want to execute STEP4 if STEP2 executes, you can code in the following way:

```
//CHE01 IF STEP2.RUN THEN  
//STEP4 EXEC MYPROC1  
// ENDIF
```

IF/THEN/ELSE/ENDIF statement construct

```
//name1    IF      relational expression THEN
.
.
.
Action when relational expression is true

//name2    ELSE
.
.
.
Action when relational expression is false

//name3    ENDIF
```

© Copyright IBM Corporation 2011

Figure 10-4. IF/THEN/ELSE/ENDIF statement construct

ES074.0

Notes:

Note the following principles regarding the IF/THEN/ELSE/ENDIF statement construct:

- The IF/THEN/ELSE/ENDIF statement construct can be placed anywhere in a job after the JOB statement.
- The IF/THEN/ELSE/ENDIF statement construct is actually three statements:
 - IF/THEN
 - ELSE
 - ENDIF
- One or more blanks must precede and follow the relational expression (to separate it from IF and THEN).
- The action consists of one or more EXEC statements and its associated JCL statements.
- The name field on the construct is optional.

- If the result is true, the action following `THEN` is taken, and execution resumes following the `ENDIF` statement.
- If the result is false, the action following `ELSE` is taken, and execution resumes following the `ENDIF` statement.
- The system will never execute both the `THEN` and the `ELSE` actions. That is, the two actions are mutually exclusive.
- The action following `THEN` can be omitted. This situation results in a null `THEN` clause. The action following `ELSE` can be omitted. This results in a null `ELSE` clause; however, both of these actions can not be omitted.
- Either the `THEN` clause or `ELSE` clause must contain at least one `EXEC` statement.

IF/THEN/ELSE/ENDIF example

```

//SAMPLE      JOB
//STEP1       EXEC     PGM=PROG1
.
.
.
//CHECK        IF      (RC > 0 | ABEND) THEN
//BADRUN      EXEC    SETFLAGS   Execute if true
//              ELSE
//GOODRUN      EXEC    PGM=ANALYZE
.
.
.
//SAVE         EXEC    PGM=SAVEDATA   Execute if false
.
.
.
//ENDCK        ENDIF
//NXSTEP       EXEC    PGM=CONTINUE

```

© Copyright IBM Corporation 2011

Figure 10-5. IF/THEN/ELSE/ENDIF example

ES074.0

Notes:

In the following example:

- An IF/THEN/ELSE/ENDIF statement construct is coded following STEP1.

```

//CHECK  IF (RC > 0 | ABEND) THEN
//BADRUN EXEC  SETFLAGS
      ....
//          ELSE
//GOODRUN EXEC PGM=ANALYZE
      ....
//SAVE    EXEC PGM=SAVEDATA
      ....
//ENDCK  ENDIF
//NXSTEP EXEC PGM=CONTINUE

```

IF STEP1 HAS A RC GREATER THAN 0 OR
 IF THE STEP ABENDS, THEN EXECUTE
 THE PROCEDURE SETFLAGS. AFTER
 SETFLAGS COMPLETES EXECUTION, THE
 CONTINUE PROGRAM WILL EXECUTE.
 IF STEP1 EXECUTES WITH A RC OF 0
 AND DOES NOT ABEND, THE PROGRAMS,
 ANALYZE AND SAVEDATA WILL EXECUTE.
 WHEN SAVEDATA COMPLETES EXECUTION,
 THE CONTINUE PROGRAM WILL EXECUTE.

JCL IF condition

```
// SET ABCLEAN='TRUE'
//COPY EXEC PGM=MYPGM
....
// IF (ABEND = &ABCLEAN | COPY.RC > 8) THEN
//FINAL EXEC PGM=CLEANUP
....
// ELSE
//CLEAN EXEC PGM=LASTPGM
....
// ENDIF
//
```

© Copyright IBM Corporation 2011

Figure 10-6. JCL IF condition

ES074.0

Notes:

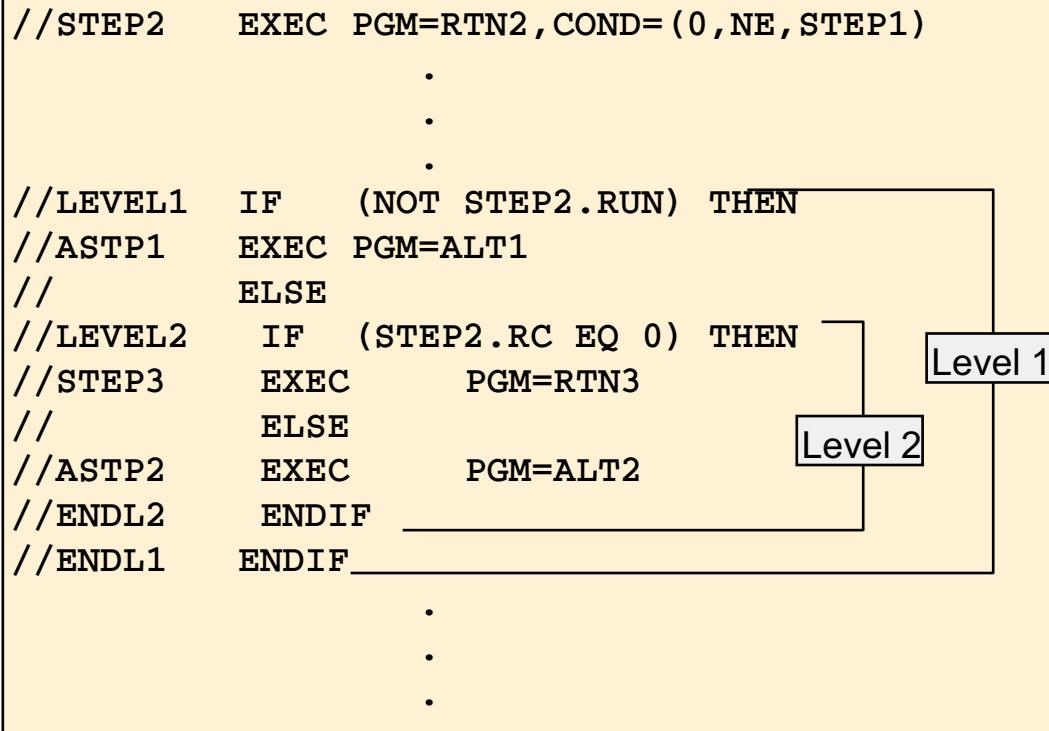
The condition portion of this statement is somewhat under-documented. It can consist of an expression which evaluates to 'TRUE' or 'FALSE' or a comparison of two values. The two values being compared must be of the same type:

- A numeric value 0-4095 (something which looks like a return code). This includes the RC operator.
- Values of the form *Uddd* where *ddd* is a decimal number 0001-4095 or *Sxxx* where *xxx* is a hexadecimal value in the range 001-FFF (something which looks like an abend code). This includes the ABENDCC operator.
- Values 'TRUE' or 'FALSE'. This includes the RUN or ABEND operator.

The operators can:

- Be specific to a step for example, ABEND.STEP1, which is set depending on whether or not STEP1 abended or
- Apply to the whole job. Did any step abend?

Nested example



© Copyright IBM Corporation 2011

Figure 10-7. Nested example

ES074.0

Notes:

This example shows that one IF/THEN/ELSE/ENDIF statement construct can be nested in another.

```

//STEP2      EXEC PGM=RTN2,COND=(0,NE,STEP1)
//LEVEL1    IF      (NOT STEP2.RUN) THEN
//ASTP1      EXEC      PGM=ALT1
//ENDL1      ENDIF
//LEVEL2    IF      (STEP2.RC EQ 0) THEN
//STEP3      EXEC      PGM=RTN3
//          ELSE
//ASTP2      EXEC      PGM=ALT2
//ENDL2      ENDIF

```

The test will cause STEP2 to be bypassed if STEP1 has a non-zero RC. If STEP 2 does not run, execute the program ALT1 and fall thru to the ENDL1 ENDIF statement.

If STEP2 executes, the system will test the STEP2 RC. If the STEP2 RC is equal to 0, the system will execute RTN3 and fall thru to the ENDL2 END IF statement. If the STEP2 RC is non-zero, the system will execute ALT2 and fall thru to the ENDL2 ENDIF statement.

SYSOUT processing: Simple form control

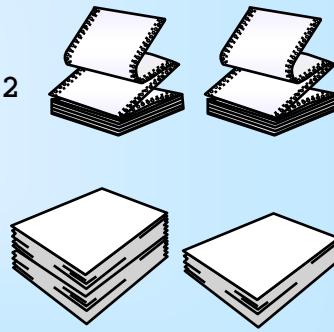
```
//ddname DD SYSOUT=(___, FORM #), COPIES=___, OUTLIM=___
```



```
//REPT1 DD SYSOUT=(A,,IPRT), COPIES=2
```



```
//REPT2 DD SYSOUT=L, OUTLIM=5000
```



© Copyright IBM Corporation 2011

Figure 10-8. SYSOUT processing: Simple form control

ES074.0

Notes:

Use the COPIES parameter to specify how many copies of this sysout data set are to be printed.

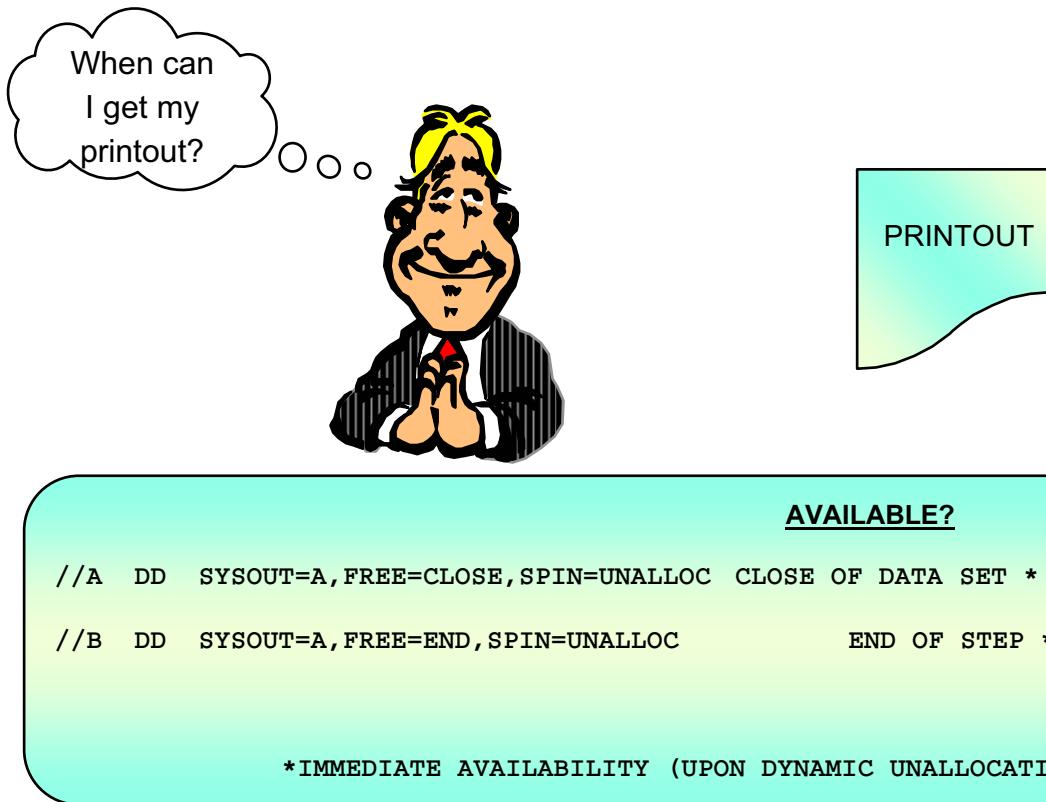
Minimum number of copies is one (default is 1).

Maximum number of copies in JES2 is 255 and, in JES3, 254.

Use the OUTLIM parameter to limit the number of logical records in the sysout data set defined by this DD statement.

The range is one to 16777215 logical records.

FREE and SPIN parameters



© Copyright IBM Corporation 2011

Figure 10-9. FREE and SPIN parameters

ES074.0

Notes:

The following principles apply to the FREE and SPIN parameters:

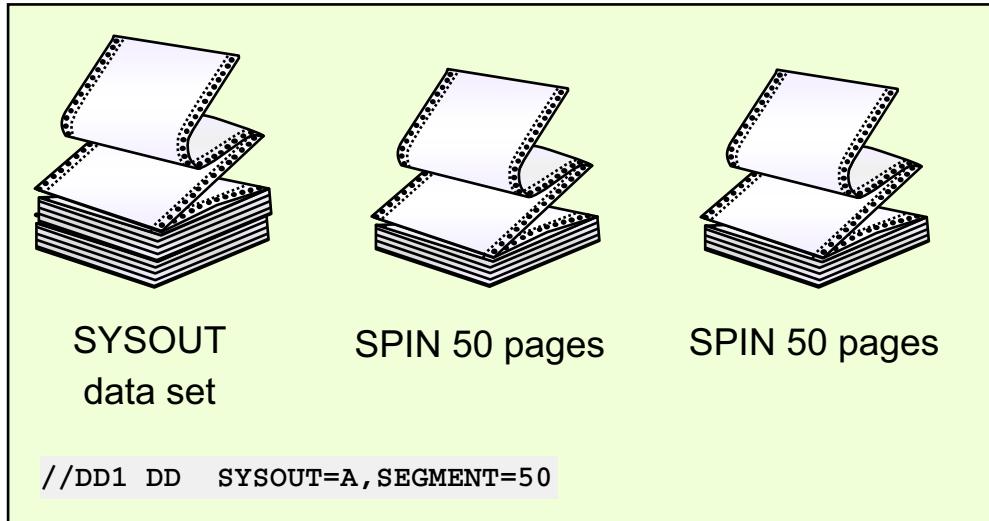
- The SPIN parameter is used to make a job's SYSOUT data available for printing at end of job.
 - SPIN=UNALLOC makes the data set available for printing when it is unallocated.
 - SPIN=NO makes the data set unavailable for printing until end of job.
- The FREE parameter is used to unallocate resources used by a data set. Resources can be devices, volumes, or exclusive use of a data set.
 - FREE=END requests that the data set be unallocated at end of step.
 - FREE=CLOSE requests that the data set be unallocated at data set close.
- The FREE parameter is more versatile since it can be used to unallocate devices that are not associated with SYSOUT output.
- FREE can be coded without SPIN. For example, coding FREE=CLOSE produces the same result as FREE=CLOSE and SPIN=UNALLOC.

- For the JESLOG= keyword on the JOB statement:
 - JESLOG=SPIN JESLOG is spin eligible.
 - JESLOG=SUPPRESS JESLOG is suppressed.
 - JESLOG=NOSPIN JESLOG is not spun.
 - JESLOG=(SPIN, *n*) JESLOG automatically spun after *n* lines in one of the data sets.
 - Where *n* is 500-999 or 1K to 999K or 1M to 999M
 - JESLOG=(SPIN, *time*) JESLOG automatically spun at time of day or time interval.
 - *Time* is *hh:mm* for time of day where *hh* = 0 to 23 and *mm* is 00 to 59
 - *Time* is *+hh:mm* for time interval where *hh* = 0 to 99 and *mm* is 00 to 59. The minimum interval is +0:10.

Examples:

//C DD SYSOUT=A, FREE=CLOSE, SPIN=NO	Available at end of job
//D DD SYSOUT=A, FREE=END, SPIN=NO	Available at end of job
//JOBNAME JOB, , JESLOG=(SPIN, '+5:13')	

SEGMENT parameter



© Copyright IBM Corporation 2011

Figure 10-10. SEGMENT parameter

ES074.0

Notes:

Note the following principles regarding the SEGMENT parameter:

- Use the SEGMENT parameter to specify the number of pages (a segment) that can be spun off and printed while the job is executing. For example, every time 50 pages are written to the DD statement in the visual, those 50 pages will be spun. Thus, the printer can be printing those 50 pages while the program is generating the next 50-page segment.
- Segments are spun off one by one and can be printed concurrently on any z/OS printers which are available.
- The print data set must also be formatted with RECFM=A or RECFM=M.
- The SEGMENT parameter overrides:
 - FREE and SPIN parameters on the DD statement.
 - OUTDISP parameter on the OUTPUT statement. (This is discussed later in the unit.)
- The SEGMENT parameter is supported in JES2 only.

SYSOUT DD statement limitations

- Only one destination is allowed in the DEST parameter.
- There is no control over the JES-created data sets for the JCL listing, the job log, or the system messages.
- Changing desired routing can require changing many parameters on several DD statements.

Solution:

The OUTPUT JCL statement

© Copyright IBM Corporation 2011

Figure 10-11. SYSOUT DD statement limitations

ES074.0

Notes:

Use the OUTPUT parameter with the SYSOUT parameter to associate a sysout data set explicitly with an OUTPUT JCL statement. JES processes the sysout data set using the options from this DD statement combined with the options from the referenced OUTPUT JCL statement.

- The SYSOUT DD statement has limitations:
 - Only one destination is allowed in the DEST parameter.
 - There is no control over the JES-created data sets for the JCL listing, the job log, or the system messages.
 - Changing desired routing can require changing many parameters on several DD statements.
- The solution is the OUTPUT JCL statement.
 - Code the OUTPUT JCL statement before any sysout DD statement that refers to it.
 - Use the OUTPUT JCL statement to specify processing options for a sysout data set. These processing options are used only when:

- The OUTPUT JCL statement is explicitly or implicitly referenced by a sysout DD statement. JES combines the options from the OUTPUT JCL statement with the options from the referencing DD statements.
 - *Explicitly* means that the sysout DD statement contains an OUTPUT parameter that specifies the name of the OUTPUT JCL statement (explicit reference).
 - *Implicitly* means that the sysout DD statement does not contain an OUTPUT statement reference (implicit reference).
- An OUTPUT JCL statement that contains a DEFAULT=YES parameter is used for an implicit reference.

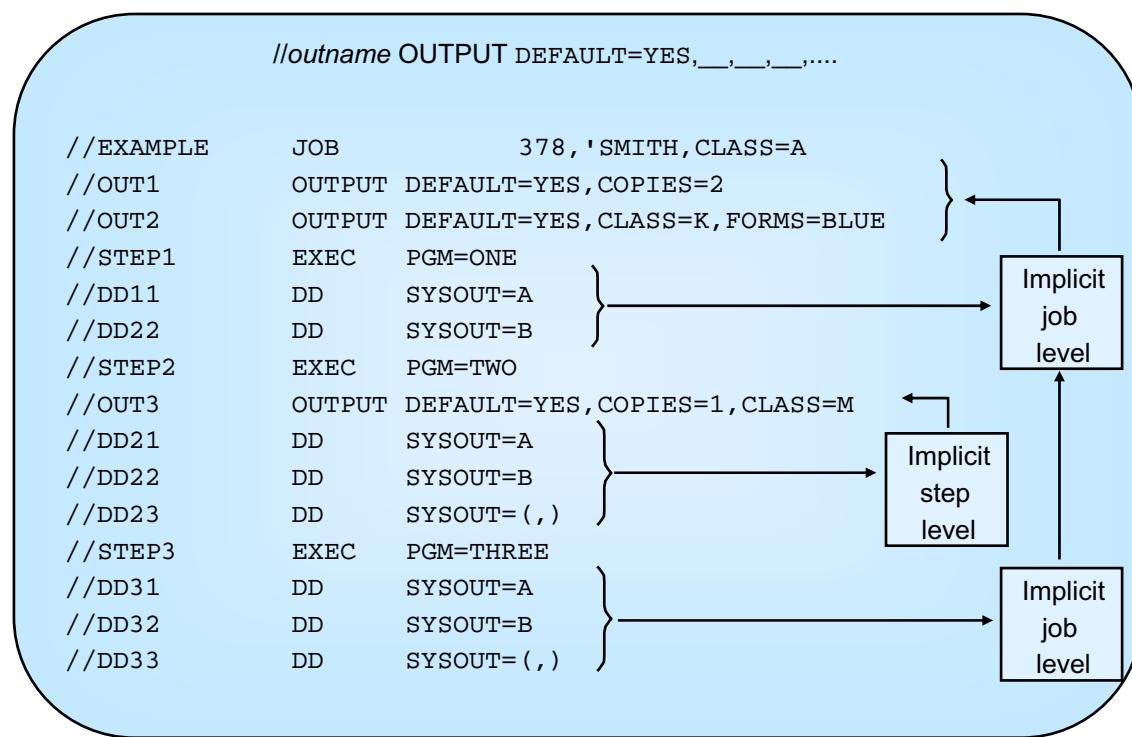
Example of explicit reference:

```
//TSOMJ01 JOB ....  
//OUT1 OUTPUT DEST=STLNODE.JDOE  
//OUT2 OUTPUT CONTROL=DOUBLE  
//STEP1 EXEC PGM=ABC  
//DS DD SYSOUT=C,OUTPUT=(*.OUT1,*.OUT2)
```

Example of step-level implicit reference:

```
//STEP2 EXEC PGM=XYZ  
//OUTC OUTPUT DEFAULT=YES,COPIES=2  
//DDABC DD SYSOUT=A
```

OUTPUT statement: Setting defaults



© Copyright IBM Corporation 2011

Figure 10-12. OUTPUT statement: Setting defaults

ES074.0

Notes:

The example in the visual shows implicit references to default OUTPUT JCL statements.

- Job-level OUTPUT JCL statements appear after the JOB statement and before the first EXEC statement.
- Step-level OUTPUT JCL statements appear in a step, that is, anywhere after the first EXEC statement in a job.

A sysout DD statement implicitly references all step-level default OUTPUT JCL statements in the same step.

A sysout DD statement implicitly references all job-level default OUTPUT JCL statements when the step containing the DD statement does not contain any step-level default OUTPUT JCL statements.

If a parameter is coded on both the DD statement and the OUTPUT statement, the parameter on the DD statement will be used.

In the example:

- DD11 and DD22 implicitly reference the job-level OUTPUT JCL statement. DD11 and DD22 will each be printed twice (OUT1) and then be printed again on blue forms (OUT2). Note that CLASS=K is ignored because class is coded on DD11 and DD22.
- DD21 implicitly references the step-level OUTPUT JCL statement OUT3 and will use class A. OUT1 and OUT2 OUTPUT statements will not be used because the step-level OUTPUT statement overrides the job-level OUTPUT statements.
- DD22 implicitly references the step-level OUTPUT JCL statement OUT3 and will print one copy using class B.
- DD23 implicitly references the step-level OUTPUT JCL statement OUT3 and will print one copy using the default class M.
- DD31 implicitly references the job-level OUTPUT JCL statement. DD31 will print the output twice (OUT1) and then be printed again on blue forms (OUT2) to class A.
- DD32 implicitly references the job-level OUTPUT JCL statement. DD32 will print the output twice (OUT1) and then be printed again on blue forms (OUT2) to class B.
- DD33 implicitly references the job-level OUTPUT JCL statement. DD33 will print the output twice (OUT1) and then be printed again on blue forms (OUT2). DD33 will use the default class K.

OUTPUT statement: Explicit reference

```
//ddname      DD  SYSOUT=___,OUTPUT=(___,___,___)

//EXPLICIT  JOB          378,SMITH
//OUT1        OUTPUT      DEST=LGA,COPIES=2
//OUT2        OUTPUT      DEST=ORD,COPIES=3
//OUT3        OUTPUT      DEFAULT=YES,CLASS=D,FORMS=GREEN
//STEP1      EXEC      PGM=ONE
//DD11        DD          SYSOUT=A,OUTPUT=(*.OUT1,*.OUT3)
//DD12        DD          SYSOUT=C
//STEP2      EXEC      PSM=TWO
//OUT7        OUTPUT      CLASS=L
//DD21        DD          SYSOUT=A,OUTPUT=*.OUT2
```

© Copyright IBM Corporation 2011

Figure 10-13. OUTPUT statement: Explicit reference

ES074.0

Notes:

DD11 explicitly references OUTPUT statements OUT1 and OUT3 to send two copies of output to LGA and one copy locally on green forms using class A.

DD12 implicitly references the default OUTPUT statement OUT3 to send one copy locally on a green form using class C.

DD21 explicitly references the OUTPUT statement OUT2 to send three copies to ORD using class A.

SYSOUT distribution parameters

```

//outname      OUTPUT NAME='__',DEPT='__',
//                  ADDRESS=( '__','__', '__', '__'),TITLE='__',
//                  BUILDING='__', ROOM='__'

//SAMPLE        JOB          791,MIKE,CLASS=A
//DEFAULT       OUTPUT DEFAULT=YES,ROOM='321'
//HOU           OUTPUT DEST=HOU,BUILDING='2A',TITLE=' POLLUTION DATA'
//ATL           OUTPUT DEST=ATL,ADDRESS=( 'L MAYS', 'ENVIROTECH',
//      '5683 WOODBURY', 'ATLANTA, GA 30320')
//TELL          OUTPUT NOTIFY=(HARPO,CHICO)
//STEP1         EXEC          PGM=PCOUNT
//DD11          DD            SYSOUT=D
//DD12          DD            SYSOUT=C,OUTPUT=*.HOU
//STEP2         EXEC          PGM=FSAVE
//DD21          DD            SYSOUT=K,OUTPUT=( *.HOU, *.ATL, *.TELL)
//DD22          DD            SYSOUT=A,OUTPUT=*.ATL

```

© Copyright IBM Corporation 2011

Figure 10-14. SYSOUT distribution parameters

ES074.0

Notes:

The following principles apply to SYSOUT distribution parameters:

- Six different OUTPUT statement parameters, ADDRESS, BUILDING, DEPT, NAME, ROOM, and TITLE, can be coded to assist in the distribution of sysout data set output.
- A single value can be coded for each parameter (1-60 text characters). Enclose value in apostrophes ('') if it contains a blank or special characters.
- ADDRESS allows up to four delivery addresses to be coded.

Four DD statements are defined in the visual:

- DD11: Class D output is printed on one part forms and is distributed locally to room 321.
- DD12: Class C output is printed in Houston. Building 2A is printed on the line reserved for building on the separator page of the output. POLLUTION DATA is printed on the line reserved for title.

- DD21: Class K output is printed in Houston and Atlanta. Building 2A and POLLUTION DATA are printed on the output for Houston. Atlanta output is addressed to:

L MAYS
ENVIROTECH
5683 WOODBURY
ATLANTA, GA 30320

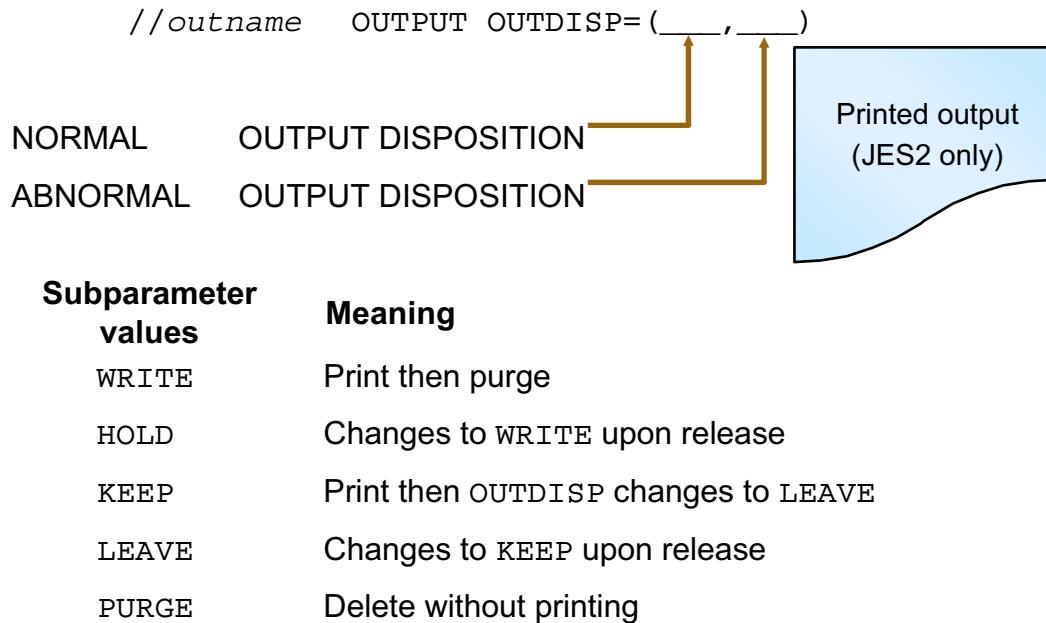
- DD22: Class A output is printed in Atlanta and is addressed to L MAYS at the address above.

The NOTIFY parameter of the TELL statement is used to inform users HARPO and CHICO that the output has been printed.

- The NOTIFY parameter can be used to inform up to four users that the system has finished printing all the data associated with this OUTPUT statement. The NOTIFY message also indicates whether the printing was successful.

NOTIFY=*ID* or NOTIFY=(*NODE.ID*,*NODE.ID*)

OUTDISP parameter



© Copyright IBM Corporation 2011

Figure 10-15. OUTDISP parameter

ES074.0

Notes:

The following principles apply to the OUTDISP parameter:

- The OUTDISP parameter indicates the disposition of a sysout data set.
- The format is:
`//OUTNAME OUTPUT OUTDISP=(normal_OUTPUT_Dispose,abnormal_OUTPUT_Dispose)`
- The five subparameters listed below can be coded for each of the two dispositions:
 - **WRITE**: The data set is to be printed then purged after printing.
 - **HOLD**: The data set is held until it is released. When released, its disposition changes to **WRITE**.
 - **KEEP**: The data set is to be printed. The disposition changes to **LEAVE** after printing is complete.
 - **LEAVE**: The data set is held until it is released. When it is released, its disposition changes to **KEEP**.
 - **PURGE**: The data set is to be deleted without printing.

Checkpoint (1 of 2)

1. True or False: The COND parameter on the first step of a job is ignored.
2. True or False: The IF construct on the first step of a job is ignored.
3. What are the constructs required after an IF?
 - a. THEN
 - b. ELSE
 - c. ENDIF
4. What is the meaning of // IF (step1. ¬ABEND?
5. True or False: You must specify the step name for an IF condition.

© Copyright IBM Corporation 2011

Figure 10-16. Checkpoint (1 of 2)

ES074.0

Notes:

Checkpoint (2 of 2)

6. True or False: If you have not given the step name, the return code from the last step is used for checking.
7. True or False: Either the THEN clause or ELSE clause must contain at least one EXEC statement.
8. True or False: The action following THEN can be omitted.
9. Give another syntax for IF (ABEND | STEP1.RC > 8).
10. True or False: If a parameter is coded on both the DD and the OUTPUT statements, the parameter on the DD statement will be ignored.

© Copyright IBM Corporation 2011

Figure 10-17. Checkpoint (2 of 2)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Describe conditional execution of job steps using the IF/THEN/ELSE/ENDIF statement construct
- Code the parameter to control the amount of output to be printed and the timing of the printing
- Use the OUTPUT statement to route SYSOUT to multiple nodes and to establish default forms control for SYSOUT data sets
- Discuss the SYSOUT distribution parameters
- Discuss the disposition of a SYSOUT data set
- Describe the NOTIFY parameter

© Copyright IBM Corporation 2011

Figure 10-18. Unit summary

ES074.0

Notes:

Unit 11. SORT and MERGE

What this unit is about

Sorting is important in data processing. There are numerous instances when substantial time can be saved by sorting records before they are analyzed or used. Some analysis or reporting programs even require that the input be sorted. Merging is just a special case of sorting. This topic discusses SORT and MERGE operations.

What you should be able to do

After completing this unit, you should be able to:

- Describe the SORT and MERGE processes
- Describe the JCL and control statements required for a SORT or MERGE operation
- Discuss JCL coding rules for SORT or MERGE job streams

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597	<i>z/OS MVS JCL Reference</i>
SC26-7527	<i>DFSORT: Getting Started</i>
SC26-7523	<i>DFSORT Application Programming Guide</i>

Unit objectives

After completing this unit, you should be able to:

- Describe the SORT and MERGE processes
- Describe the JCL and control statements required for a SORT or MERGE operation
- Discuss JCL coding rules for SORT or MERGE job streams

© Copyright IBM Corporation 2011

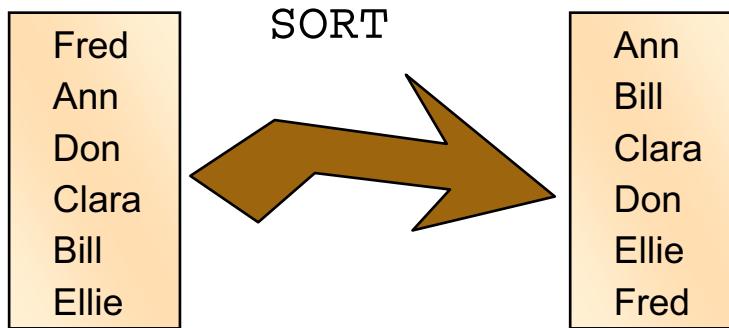
Figure 11-1. Unit objectives

ES074.0

Notes:

SORT process

- **SORT:** Arrange logical records in ascending or descending order.



© Copyright IBM Corporation 2011

Figure 11-2. SORT process

ES074.0

Notes:

The IBM program DFSORT provides three data processing functions:

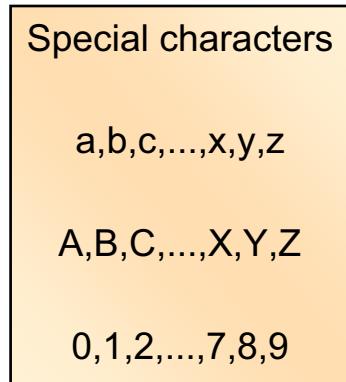
- It sorts records. Sorting is a process of arranging records into a particular preferred sequence. This sequence usually differs from the original sequence.
- It merges records. Merging is a process of combining multiple presorted data sets into a single sorted output data set.
- It copies records. DFSORT provides a more efficient copy function, ICEGENER, which is an alternative to IEBGENER. ICEGENER can be used anywhere IEBGENER is used. If an installation attempts to invoke ICEGENER in an illegal environment, control is automatically passed to IEBGENER.

DFSORt has utilities such as ICETOOL which is a multipurpose DFSORT utility that uses the capabilities of DFSORT to perform multiple operations on one or more data sets in a single step.

In the example in the visual, names are alphabetized (that is, sorted into ascending order).

EBCDIC collating sequence

- **Collating sequence:** An ordering assigned to a set of items



© Copyright IBM Corporation 2011

Figure 11-3. EBCDIC collating sequence

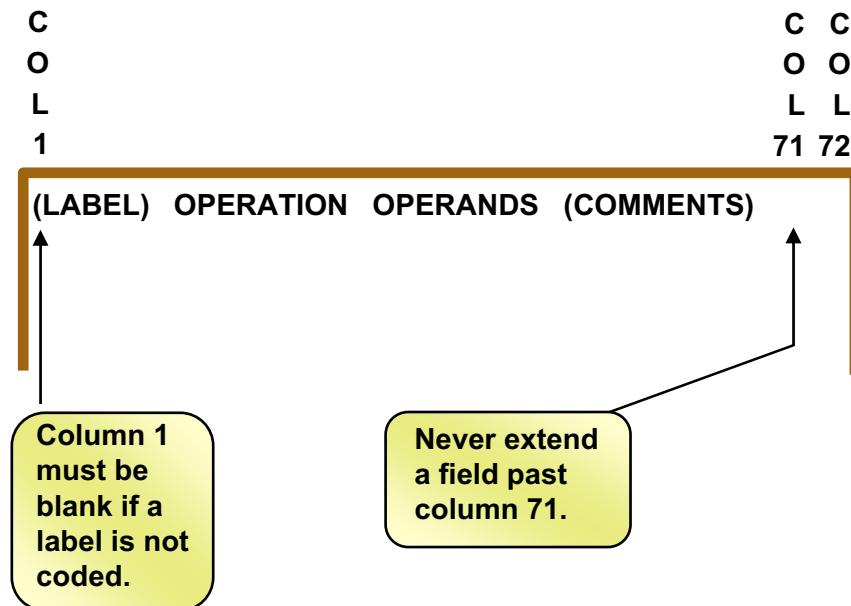
ES074.0

Notes:

The two most common collating sequences in data processing are:

- Extended Binary Coded Decimal Interchange Code (EBCDIC)
- International/American National Standard Code for Information Interchange (ASCII)

Control statement format



© Copyright IBM Corporation 2011

Figure 11-4. Control statement format

ES074.0

Notes:

The DFSORT control statements contain four fields:

- LABEL
- OPERATION
- OPERANDS
- COMMENTS

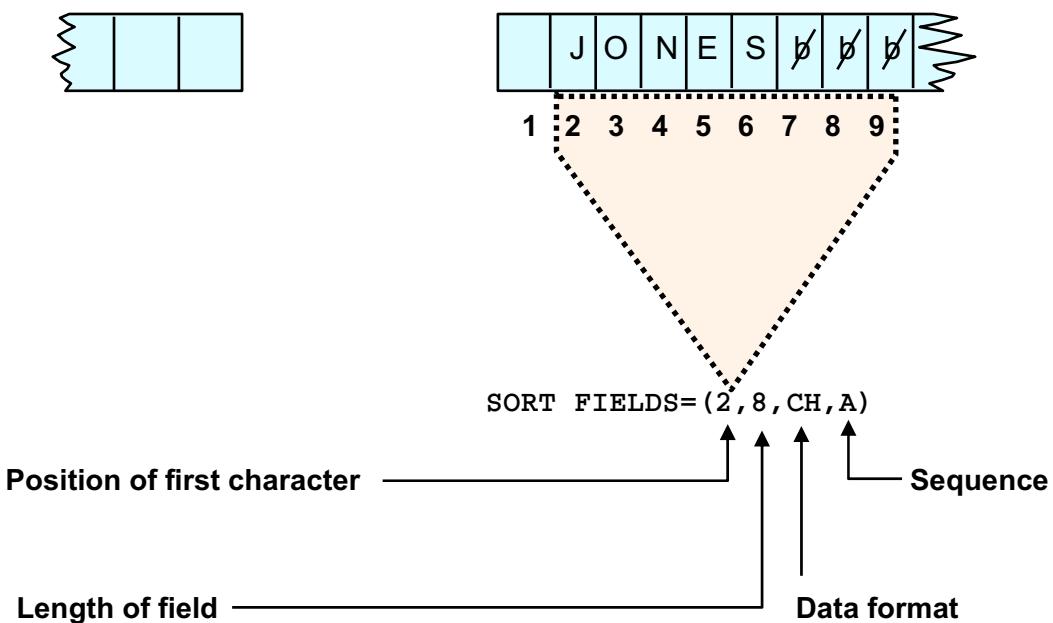
All fields must be separated by one or more blanks.

The LABEL and COMMENT fields are optional.

Continuation rules are as follows:

- To continue an operands field to the following line, break the field after a comma and leave column 72 blank.
- Continue the operand field on the next line in any column from 2 through 71.

SORT control statement



© Copyright IBM Corporation 2011

Figure 11-5. SORT control statement

ES074.0

Notes:

This sort control statement causes records to be sorted in ascending order on a single field.

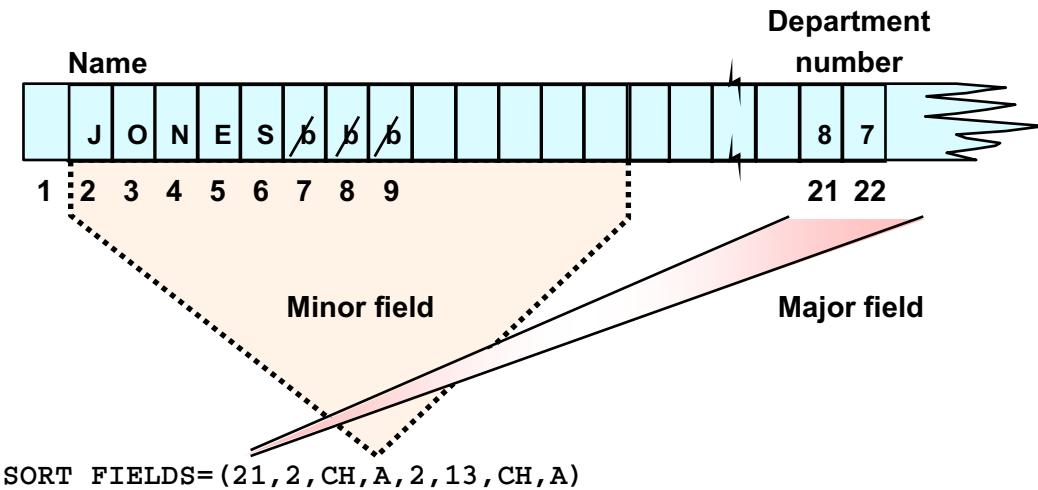
The format of the sort field is:

`SORT FIELDS=(2,8,CH,A)`

Position _____				Sequence to Sort ('A' scending/'D' ecending)
of Field				
Length _____				Data Format
of Field				

Beginning in byte 2 of each record in the input data set, for a length of 8 bytes, character data is sorted in ascending sequence.

SORT on two fields



© Copyright IBM Corporation 2011

Figure 11-6. SORT on two fields

ES074.0

Notes:

You can sort on more than one field.

- The example in the visual invokes a sort by department and then by name within department. The major field (primary field) is coded first and then the minor field (secondary field). The major sort field is always identified by coding it first in the FIELDS operand.
- If all fields have the same data format, the sort control statement can be written in the following form:

```
SORT FIELDS=(21,2,A,2,13,A),FORMAT=CH
```

SORT JCL statements

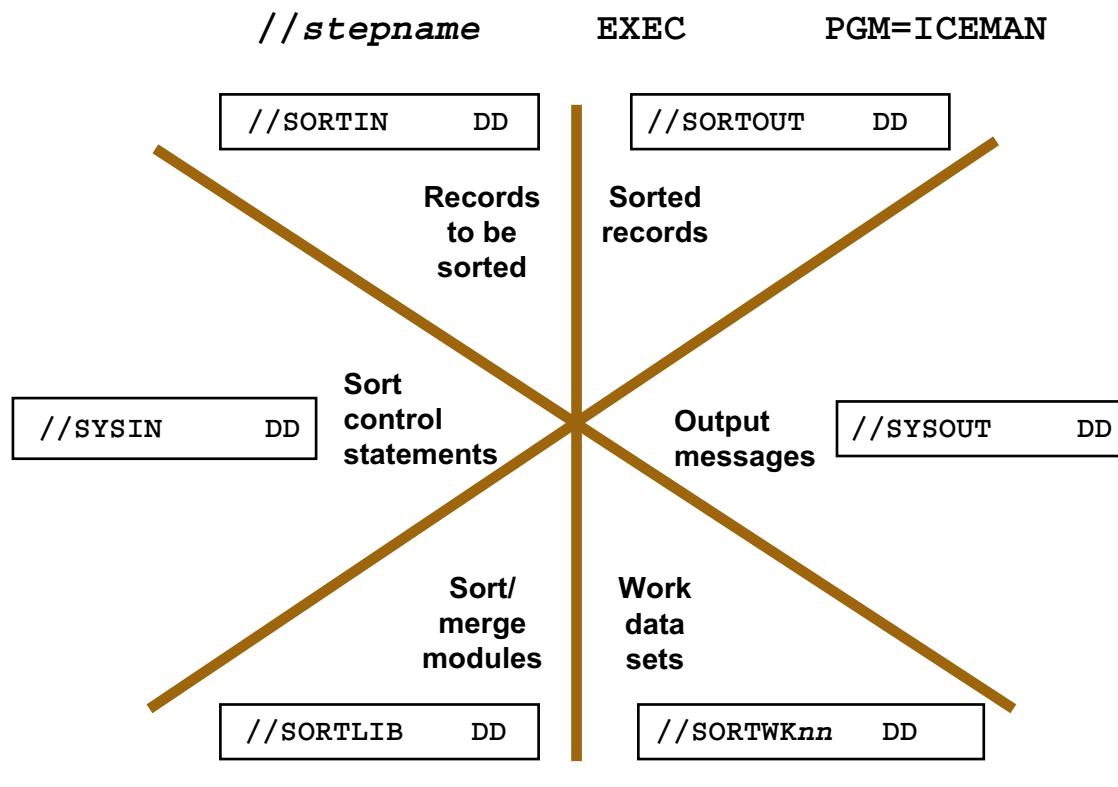


Figure 11-7. SORT JCL statements

ES074.0

Notes:

These are the major sort JCL statements.

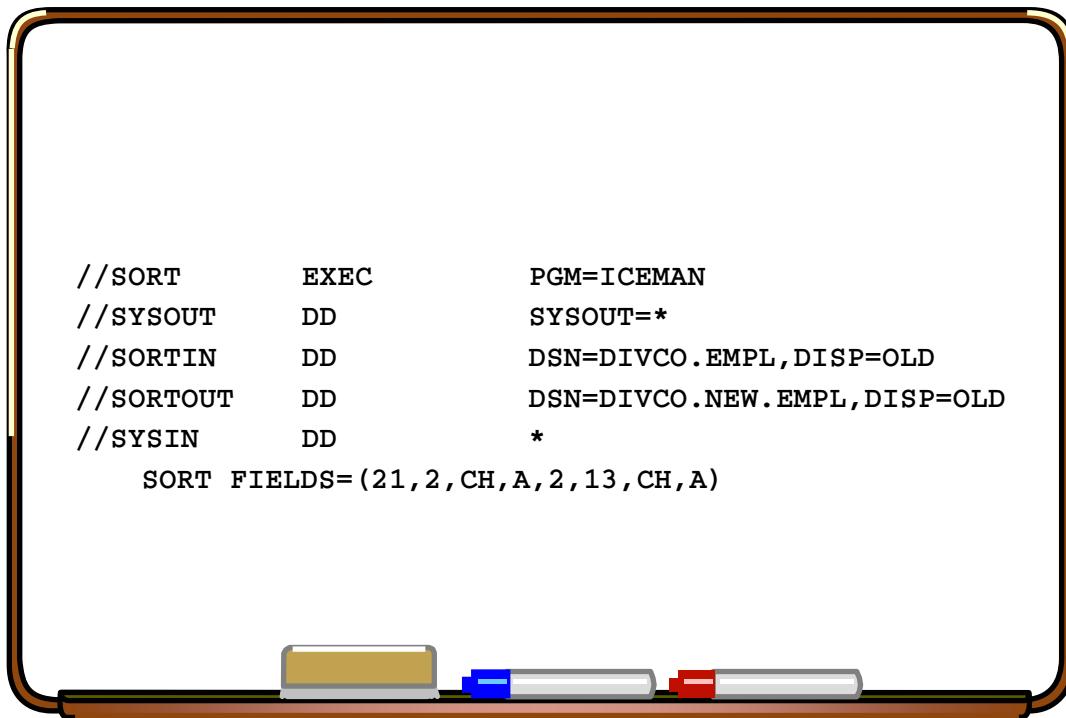
PGM=SORT or PGM=ICEMAN on the EXEC statement will invoke DFSORT. Actually, SORT is an alias or alternate name for ICEMAN.

Any valid ddname of the form SORTWK dd or SORTWK d can be used for work data sets. Of course, SORTWK00-99 and SORTWK0-9 can still be used, but so can ddnames like SORTWK3B, SORTWK#5, SORTWKXY and SORTWKZ. Up to 255 work data sets can be used in one step.

DFSORT can use multiple hiperspaces to take advantage of hipersorting for larger sorts.

z/OS DFSORT can exploit 64-bit real architecture by using memory objects for sort applications, when appropriate. A memory object is a data area in virtual storage that is allocated above the bar and backed by central storage. When a memory object is used, hiperspace and data space are not needed.

SORT JOB example



© Copyright IBM Corporation 2011

Figure 11-8. SORT JOB example

ES074.0

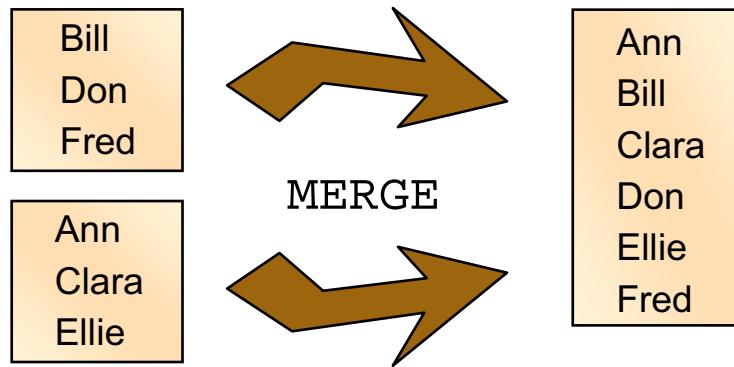
Notes:

This sort job reads new employee records from the input data set, DIVCO.EMPL, and places the sorted records in the output data set, DIVCO.NEW.EMPL.

- The EXEC statement invokes the program.
- The SYSOUT DD statement defines the message data set.
- The SORTIN DD statement defines the input data set.
- The SORTOUT DD statement defines the output data set.
- The SYSIN DD statement defines the control data set.
- The SORT control statement specifies two sort fields.

MERGE process

- **MERGE:** Combine sorted files into a single sorted file.



© Copyright IBM Corporation 2011

Figure 11-9. MERGE process

ES074.0

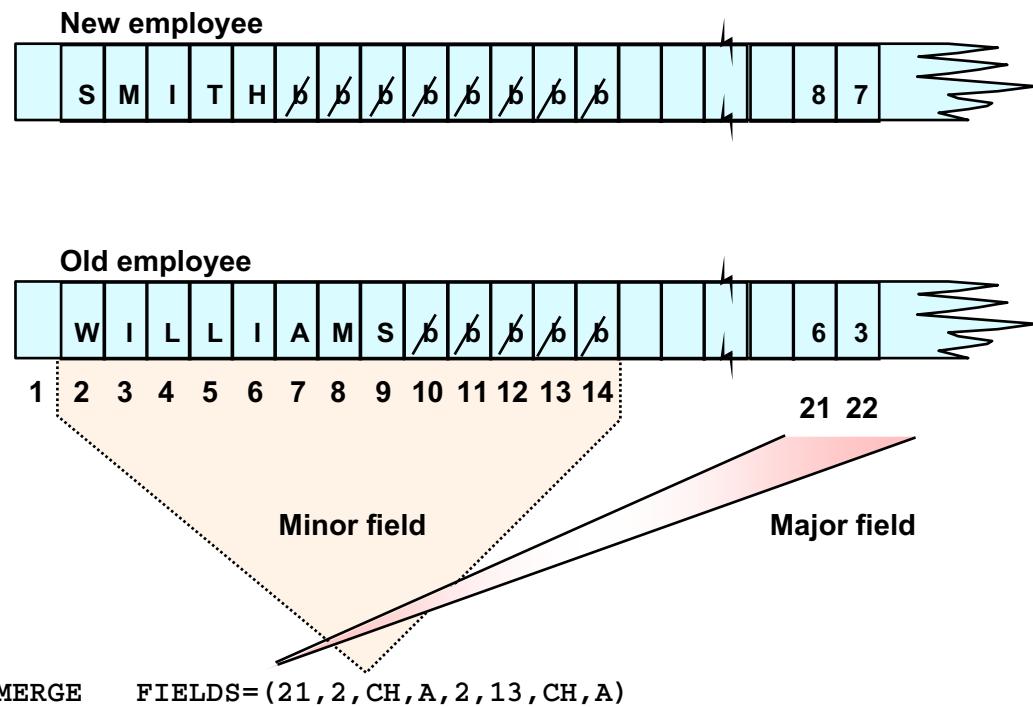
Notes:

This process takes records from up to 100 sorted data sets and combines them in a single sorted output data set.

Each of the input data sets must be previously sorted in the same sequence before the merge.

In the example in the visual, two sorted data sets are merged into a single sorted data set.

MERGE control statement



© Copyright IBM Corporation 2011

Figure 11-10. MERGE control statement

ES074.0

Notes:

The `MERGE` control statement format is similar to that of the sort.

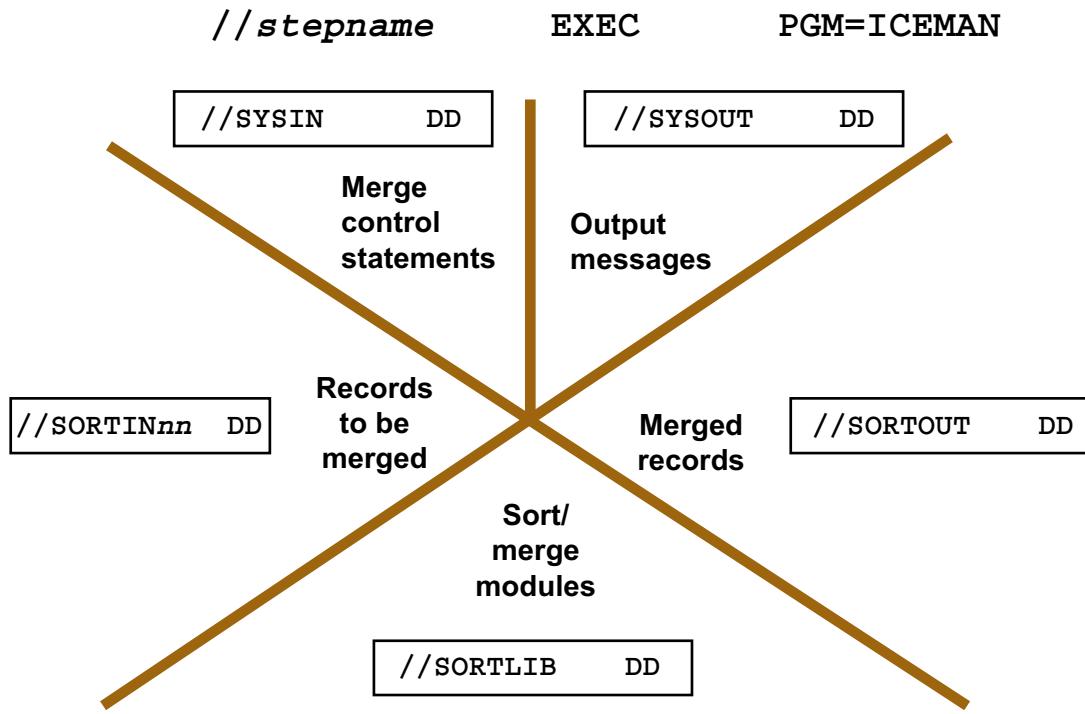
Input data sets must already be sorted in the same sequence.

If all fields have the same format, the `MERGE` control statement can be written in the form:

```
MERGE FIELDS=(21,2,A,2,13,A),FORMAT=CH
```

In this example, new employee records are being merged with old employee records.

MERGE JCL statements



© Copyright IBM Corporation 2011

Figure 11-11. MERGE JCL statements

ES074.0

Notes:

These are the major MERGE JCL statements. They are used in almost every merge job.

The `SORTWKnn DD` statements are not used by the MERGE process.

`PGM=SORT` or `PGM=ICEMAN` on the `EXEC` statement will invoke DFSORT. Actually, `SORT` is an alias or alternate name for `ICEMAN`.

MERGE JCL example

```
//MERGE          EXEC   PGM=ICEMAN
//SYSOUT         DD     SYSOUT=*
//SORTIN01        DD     DSN=DIVCO.NEW.EMPL,DISP=OLD
//SORTIN02        DD     DSN=DIVCO.OLD.EMPL,DISP=OLD
//SORTOUT         DD     DSN=DIVCO.TOT.EMPL,DISP=OLD
//SYSIN          DD     *
      SORT FIELDS=(21,2,CH,A,2,13,CH,A)
```

© Copyright IBM Corporation 2011

Figure 11-12. MERGE JCL example

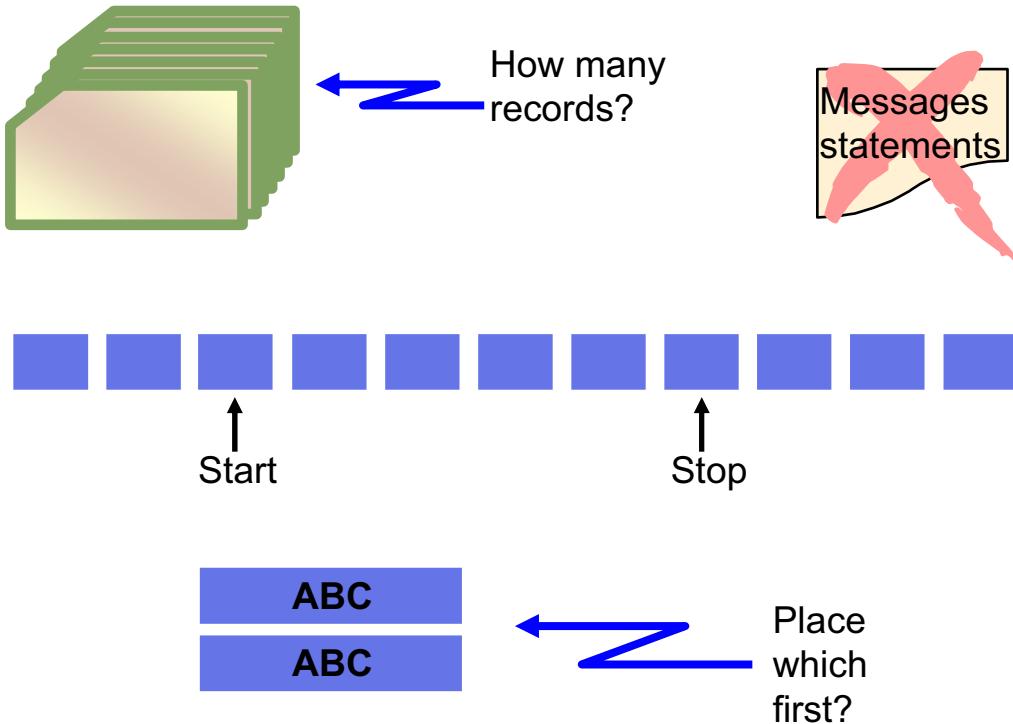
ES074.0

Notes:

This job merges new employee records in the input data sets, DIVCO.NEW.EMPL and DIVCO.OLD.EMPL.

- The sorted output is placed in the output data set, DIVCO.TOT.EMPL.
- The EXEC statement invokes the program.
- The SYSOUT DD statement defines the message data set.
- The SORTIN01 DD statement defines an input data set.
- The SORTIN02 DD statement defines an input data set.
- The SORTOUT DD statement defines the output data set.
- The SYSIN DD statement defines the control data set.
- The MERGE control statement specifies two merge fields.

SORT/MERGE considerations



© Copyright IBM Corporation 2011

Figure 11-13. SORT/MERGE considerations

ES074.0

Notes:

Additional operands can be coded on the SORT, MERGE, or OPTION statements that provide added flexibility and improve the efficiency of DFSORT. Some of those operands are:

- FILSZ=E_____ specifies the estimated number of records in the input.
- FILSZ=_____ specifies the total number of records in the input.
- SKIPREC=_____ specifies how many records should be skipped at the beginning of the input data set.
- STOPAFT=_____ specifies how many records should be read and sorted.

If any input records have identical sort/merge fields, EQUALS tells DFSORT to preserve their order on output.

A number of techniques can be used to improve sort performance. A few suggestions are given below.

- Use large blocksizes for input and output data sets.
- Accurately specify or estimate the size of the input data set using the FILSZ parameter.
- Use a large region size on the sort job step.

- Avoid especially active devices when selecting sort work devices.
- Allocate sort work space in cylinders rather than tracks or blocks.
- Select the same size for each sort work data set.
- Use multiple sort work data sets.
- Pick the same device type for all the sort work data sets.
- Use a hipersort instead of using sort work data sets.
- Use memory object sorting, a DFSORT capability that uses a memory object on 64-bit real architecture to improve the performance of sort applications.
- Avoid the use of tape sorts.
- Use ICEGENER instead of IEBGENER (for copying applications).

Example:

```
OPTION FILSZ=E10000,SKIPREC=3025,  
STOPAFT=8000
```

Another SORT example

```
//SORT      EXEC  PGM=ICEMAN
//SYSOUT    DD     SYSOUT=*
//SORTIN    DD     DSN=DIVCO.EMPL,DISP=OLD
//SORTOUT   DD     DSN=DIVCO.NEW.EMPL,DISP=OLD
//SYSIN    DD     *
      SORT FIELDS=(21,2,A,2,13,A),FORMAT=CH
      OPTION FILSZ=E75000
/*
```

© Copyright IBM Corporation 2011

Figure 11-14. Another SORT example

ES074.0

Notes:

This job will sort an estimated 75,000 records in ascending order on two fields.

Both fields have character format.

If any input records have identical sort fields, the order of the records will be preserved on output.

The OPTION control statement can also be used for merge operations.

Checkpoint (1 of 2)

1. Which of the following can DFSORT perform:
 - a. Sort records
 - b. Merge records
 - c. Copy records
 - d. All of the above
2. True or False: Fields are sorted from right to left as they appear in the FIELDS operand.
3. What is the maximum number of SORTWK data sets?
4. Name three types of SORTWK intermediate work space that DFSORT can use for sorting.
5. What is the maximum number input of data sets that can be merged by SORT?

© Copyright IBM Corporation 2011

Figure 11-15. Checkpoint (1 of 2)

ES074.0

Notes:

Checkpoint (2 of 2)

6. What is the name of the input DD statement used for merge?

7. Explain the following sort job:

```
//SORTIT EXEC PGM=SORT,PARM='ABEND'  
//SORTIN DD * number in column 1, name in 10, state in 19  
9283873    JOAN      CT  
7023232    JANE      CN  
8432343    MARIO     MA  
5549023    JILL      CT  
6998781    JENNIFER VT  
/*  
//SORTOUT DD SYSOUT=*  
//SYSIN   DD *  
          SORT FIELDS=COPY,SKIPREC=3,STOPAFT=2
```

© Copyright IBM Corporation 2011

Figure 11-16. Checkpoint (2 of 2)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Describe the SORT and MERGE processes
- Describe the JCL and control statements required for a SORT or MERGE operation
- Discuss JCL coding rules for SORT or MERGE job streams

© Copyright IBM Corporation 2011

Figure 11-17. Unit summary

ES074.0

Notes:

Appendix A contains many SORT JCL jobs that you can refer to for many different purposes.

Unit 12. Multivolume and tape allocation

What this unit is about

Some data sets are too large for a single volume, and multiple DASD or tape volumes are required to hold the data set. Instances also arise where it is advantageous to use multiple devices when accessing the data set. This topic discusses the considerations involved in coding and handling multivolume and multi-device data sets.

What you should be able to do

After completing this unit, you should be able to:

- Code DD statements to allocate a multivolume DASD data set
- Describe the characteristics of IBM standard tape labels
- Code tape label processing options on the DD statement
- Code DD statements to allocate and locate a multivolume tape data set using single and multiple tape units
- Discuss the types of tape drives and methods of requesting them
- Explain the differences among tape label types
- Code DD statements utilizing the UNIT=AFF=DDNAME parameter to reduce device requirements

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597	<i>z/OS MVS JCL Reference</i>
SA22-7598	<i>z/OS MVS JCL User's Guide</i>
SC26-4923	<i>DFSMS/dfp Using Magnetic Tape Labels and File Structure</i>

Unit objectives

After completing this unit, you should be able to:

- Code DD statements to allocate a multivolume DASD data set
- Describe the characteristics of IBM standard tape labels
- Code tape label processing options on the DD statement
- Code DD statements to allocate and locate a multivolume tape data set using single and multiple tape units
- Discuss the types of tape drives and methods of requesting them
- Explain the differences among tape label types
- Code DD statements utilizing the UNIT=AFF=DDNAME parameter to reduce device requirements

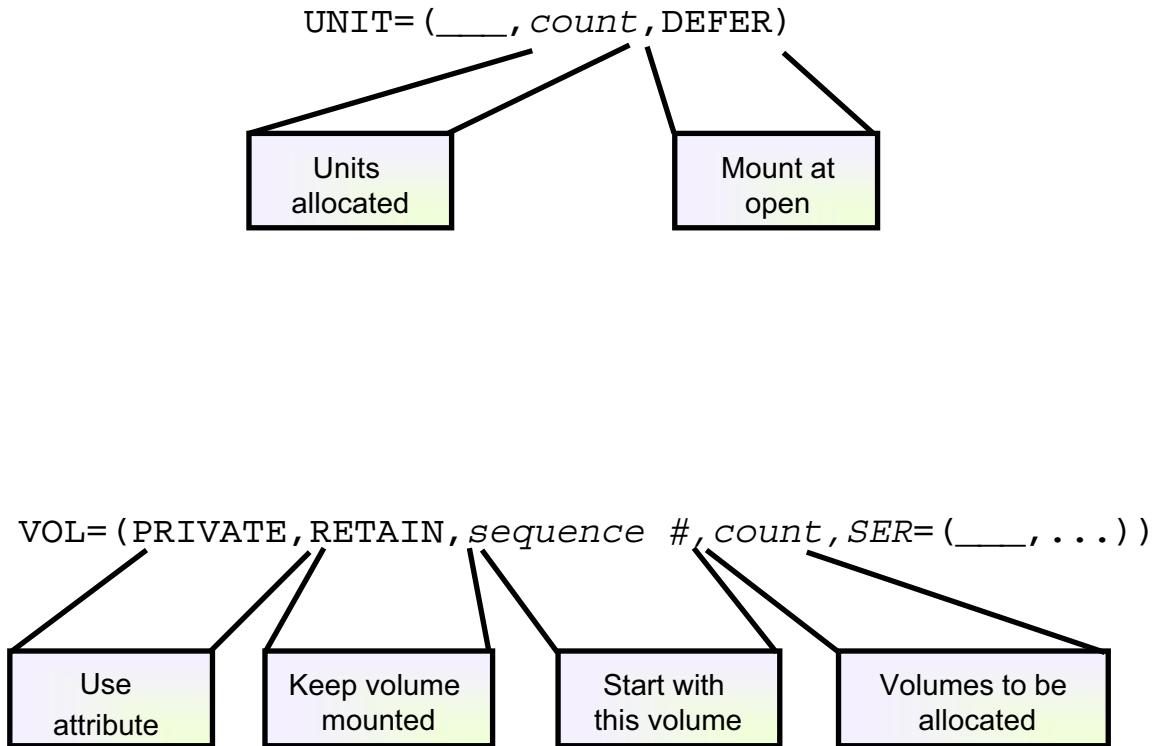
© Copyright IBM Corporation 2011

Figure 12-1. Unit objectives

ES074.0

Notes:

Unit and volume syntax



© Copyright IBM Corporation 2011

Figure 12-2. Unit and volume syntax

ES074.0

Notes:

UNIT requests a device or group of devices. **UNIT** has three subparameters.

- The first subparameter requests a device (**UNIT=3390**) or group of devices (**UNIT=SYSDA** or perhaps **UNIT=TAPE**).
- The second subparameter requests a specific number of devices. This subparameter is honored for non-SMS managed DASD and tape. It is honored for SMS tape data sets. It is ignored for SMS DASD.
- **DEFER** requests that the volume not be mounted until data set OPEN.

VOL identifies a volume or volumes on which a data set resides or will reside. **VOL** has five subparameters.

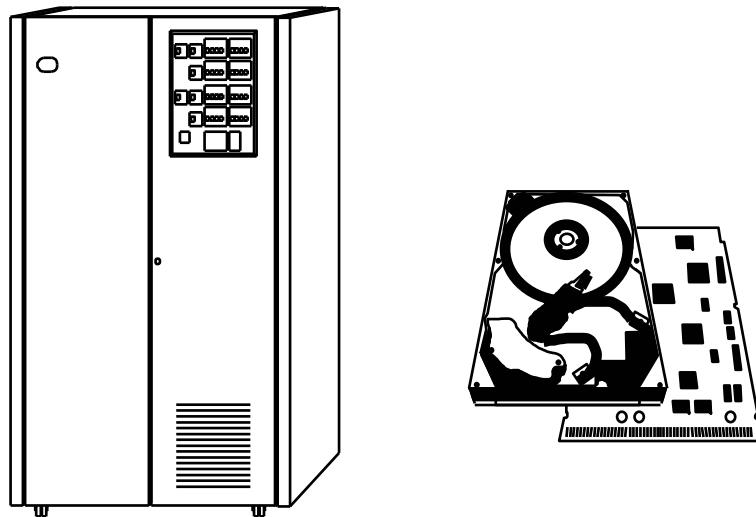
- The subparameter **PRIVATE** requests a private volume. (A *private volume* is one that is available only if it is specifically requested.)
- The subparameter **RETAIN** is used only for private tape volumes. **RETAIN** specifies that a volume is not to be demounted at data set close or end of step.

- The sequence number identifies the volume (in an *existing* multivolume data set) on which processing is to begin.
- Count specifies the maximum number of volumes for an output data set.
- SER identifies the volumes on which the data set resides or will reside.

Volume serial numbers in the VOL parameter are ignored when SMS-managed data sets are *created*. (Exception: If GUARANTEED_SPACE=YES is specified in the storage class construct, SMS will honor the coding of volume serial numbers.)

Volume serial numbers in the VOL parameter are ignored when the system locates *existing* SMS data sets. The catalog is used to locate the data set.

Multivolume DASD



© Copyright IBM Corporation 2011

Figure 12-3. Multivolume DASD

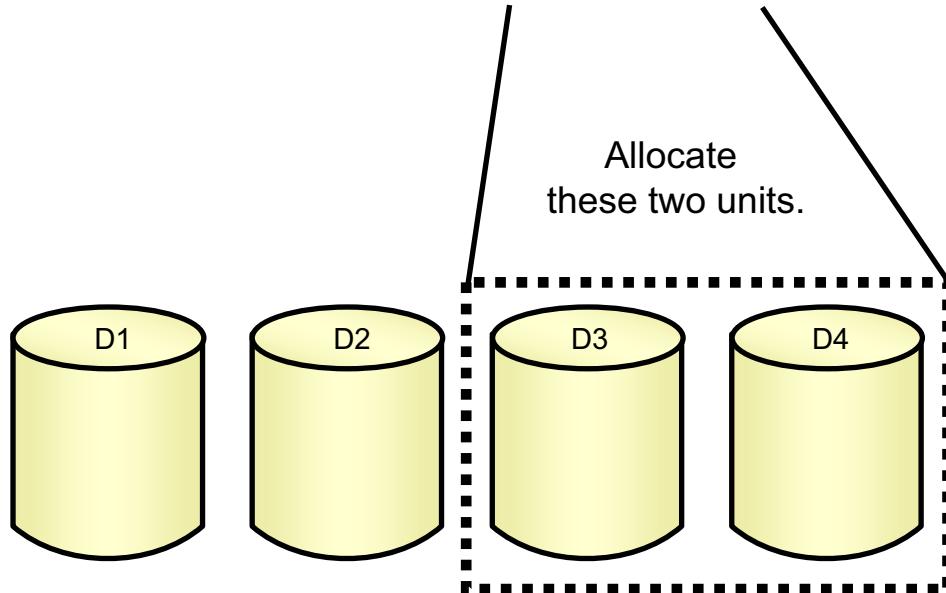
ES074.0

Notes:

The first portion of this unit discusses DASD processing only.

Multivolume: Example

```
//MVDD    DD   DSN=A.B.C,DISP=(,CATLG),UNIT=(SYSDA,2),
//                                         SPACE=(CYL,(100)),VOL=SER=(D3,D4)
```



© Copyright IBM Corporation 2011

Figure 12-4. Multivolume: Example

ES074.0

Notes:

The coder requested two units and two volumes, D3 and D4. The system uses D3 and D4 if DSN=A.B.C is not SMS-managed. If DSN=A.B.C is SMS-managed, the system picks the volumes. In this case, it is unlikely SMS will pick D3 and D4. Assume DSN=A.B.C is not SMS-managed.

- This is an example of a *specific* volume request; one in which VOL=SER has been coded or implied on the DD statement. (If a data set is cataloged, the coding is implied.)
- A *nonspecific* volume request is one in which VOL=SER has been omitted (and not implied) on the DD statement.



Questions

How would you change this example to request all four volumes in the non-SMS case?

If the data set is processed at a later point using //MVDD DD DSN=A.B.C,DISP=OLD will you need to know how many volume serial numbers are in the catalog?

Multivolume: Basic rules

- If a count and serial numbers appear in a volume parm:

```
//MVDD DD _____, _____, VOL=( , , , 5, SER=(V1, V2) )
```

- The larger of the two is used.

- If unit count and volume count disagree:

```
//MVDD DD _____, UNIT=(SYSDA, 2), VOL=( , , , 5)
```

- The number of units is allocated.
 - Specified in unit count if UNIT=(____, P) is not coded.
 - Specified by volume if UNIT=(____, P) is coded.

© Copyright IBM Corporation 2011

Figure 12-5. Multivolume: Basic rules

ES074.0

Notes:

These rules apply only to non-SMS managed data sets. SMS makes unit count and volume decisions for the data sets it manages.

If both a volume count and volume serial information are coded, the system uses the larger of the two as the volume count. The system treats V1 and V2 as specific volume requests. Then, the system will allow up to three additional nonspecific volumes to be requested (for a total of 5).

The unit count can be either a numeric value, or P can be coded.

- If the unit count is numeric for a nonspecific volume request, the system determines how many units to allocate by using the unit count.
- If the unit count is numeric for a specific, non-demountable volume request, the system determines how many units to allocate by using the volume count.
- If the unit count specifies P, the system determines the number of units by setting it equal to the larger of:
 - The volume count in the VOL parameter
 - The count of serial numbers in the SER subparameter of VOLUME

The value P in the UNIT parameter means mount in parallel.

Multivolume allocation

- How many units and volumes are allocated?

```
//PR1 DD __,UNIT=3390,VOL=SER=(VOL1,VOL2,VOL3)
//PR2 DD __,UNIT=(3390,P),VOL=SER=(VOL1,VOL2,VOL3)
```

```
//PR3 DD __,UNIT=(3390,2),VOL=(,,,4),SPACE=__
```

```
//PR4 DD DSN=A.B.C,DISP=OLD
```

```
//PR5 DD __,UNIT=(3390,P),VOL=(,,,4),SER=(VOL1,VOL2) )
```

© Copyright IBM Corporation 2011

Figure 12-6. Multivolume allocation

ES074.0

Notes:

Assume that the data sets are not SMS-managed.

- **PR1:** This is a specific volume request in which three non-demountable volumes are requested. The system allocates three units and uses three volumes.
- **PR2:** The *P* adds nothing to this DD statement. Three non-demountable volumes must necessarily be mounted on three units. This is the same as PR1.
- **PR3:** The unit count is 2. The volume count is 4. This is a nonspecific volume request. The system uses two units and allows two volumes for the creation of the data set. (The unit count overrides the volume count for nonspecific, non-demountable requests unless UNIT=(____, P) is coded.)
- **PR4:** The system checks the catalog and allocates the units and volumes accordingly.
- **PR5:** The system will use 4 as the volume count. The UNIT parameter specifies *P*, therefore the system will allocate four units and use four volumes.

Multivolume: Allocation examples

```
//BIG1 DD DSN=BIG.ONE,DISP=(,CATLG),UNIT=(3390,P),VOL=(, , , 3),
//           SPACE=(CYL,(400,200))

//BIG2 DD      DSN=BIG.TWO,DISP=OLD,UNIT=(,P)

//BIG3 DD DSN=BIG.THREE,DISP=OLD,UNIT=(,P),VOL=(, , ,3),
//           SPACE=(CYL,(200,100))
```

- Questions:

- Why was parallel mount requested in BIG1?
- Is parallel mount really needed in BIG2?
- Why is the SPACE parameter coded in BIG3?

© Copyright IBM Corporation 2011

Figure 12-7. Multivolume: Allocation examples

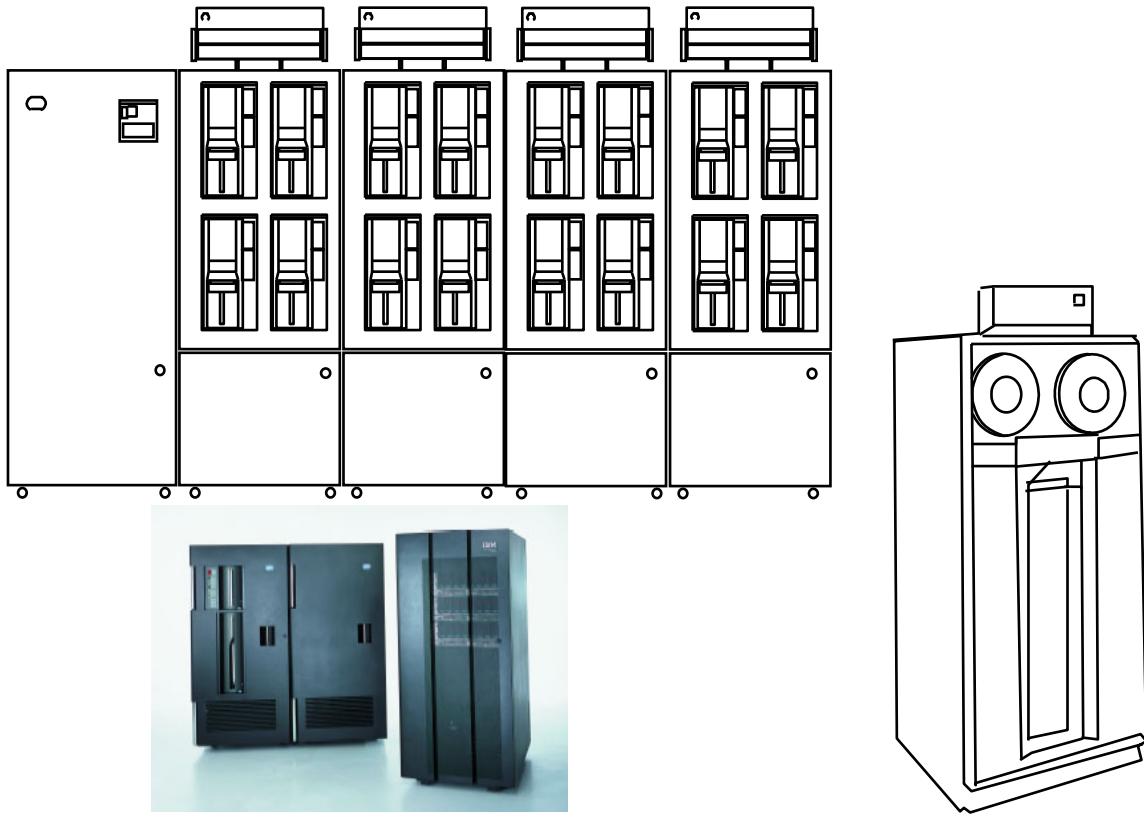
ES074.0

Notes:

Assume that the data sets are not SMS-managed.

- **BIG1:** Parallel mount is requested in BIG1 because the default number of units is 1. The unit count overrides the volume count for nonspecific requests. Thus to get more than one nonspecific volume, the coder had to specify P for the unit count.
- **BIG2:** It is not needed. The data set is cataloged, so the catalog implicitly determines the volume count. The same number of units will be used. (If this is *mountable* DASD, the answer is *yes!*)
- **BIG3:** Since UNIT=(,P) is coded together with VOL=(,,3), three units and three volumes will be assigned. It can be used to extend secondary extents to additional volumes (even if secondary extents were not originally specified); however, the primary extent coded on this DD statement must be the same as the initial primary extent.

Tape processing



© Copyright IBM Corporation 2011

Figure 12-8. Tape processing

ES074.0

Notes:

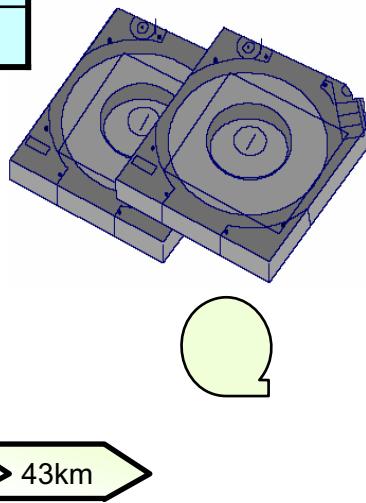
The remaining portion of this topic discusses tape processing only.

The tape environment

Possibilities:

	SMS-managed	Non-SMS-managed
DASD	Yes	Yes
TAPE	Yes	Yes

- Also for tapes:
 - Cartridges and reels
 - Automated mounts and demounts
 - Multiple track formats
 - Maximum drive-host distances
 - Wide variation in tape capacities



© Copyright IBM Corporation 2011

Figure 12-9. The tape environment

ES074.0

Notes:

Tape storage is widely used for archival and long-term storage of data in the storage hierarchy. Some shops still use round tape 3420 devices, but most have moved to 3480/3490/3590-style square tapes and beyond. Many clients use tape libraries that automate loading of tape cartridges from a large self-contained tape library. Also, boxes such as the Virtual Tape Server (VTS) not only automate tape operations but also help provide more efficient use of tape storage and speed up operations to tape through caching technology.

DASD and tape data sets can be SMS-managed or non-SMS-managed. The Removable Media Manager (DFSMSSrmm), can be used to manage both SMS-managed and non-SMS-managed tapes. In addition, DFSMSSrmm:

- Interfaces with the 3494 and 3495 Tape Library Dataservers and VTS
 - These data servers support libraries of SMS-managed tape cartridges. The libraries consist of racks (in a bookcase arrangement) where cartridges can be stored.

- The data servers automate the retrieval, mounting, demounting, and storage of tape cartridges to and from these libraries and drives of the 3490, 3490E, and 3590 tape subsystems. Operator intervention is not required.
- Manages private and scratch tapes and monitors changes to these two pools
- Interfaces with RACF to provide authorization and security for tapes
- Allows reports to be generated about the tape inventory
- Supports other functions

The 3590 technology supports data transfer rates of 9 MBps, drive-channel host distances of up to 43 km, and rewind times of two seconds.

The IBM 3592 TotalStorage Enterprise Tape Drive supports data transfer rates of up to 40 MBps (without compression), and up to 300 GB per cartridge (without compression).

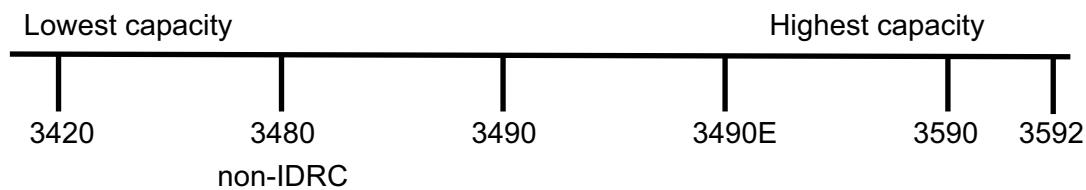
The 3592 system is the successor of the IBM Magstar 3590 family of tape drives and controller types.

The IBM 3592 tape drive can be used as a standalone solution or as an automated solution within a 3494 tape library.

Tape capacity

- To increase capacity, use:
 - Large BLKSIZE
 - Cartridges not reels
 - Data compaction

//DD1 DD TRTCH=COMP,UNIT= { 3590
3590-1
3490
3480X } . . .



© Copyright IBM Corporation 2011

Figure 12-10. Tape capacity

ES074.0

Notes:

The following information pertains to tape capacity:

- The *capacity* of a tape refers to the amount of data that can be stored on a tape.
- Several factors influence tape capacity:
 - Recording medium
 - Type of drive
 - Blocksize
 - Interblock gaps
 - Recording density
 - Data compaction
- The 3590, 3490E, 3490, and 3480 drives use cartridges. The older 3420 and 3430 drives use reels.

- Capacity limits for various types of drives are:
 - 120 MB for 3420 drives
 - 200 MB for 3480 Cartridge System Tape (CST) drives
 - 600 MB for improved data-recording capability (IDRC) drives
 - 1200 MB for 3490E (36-track bidirectional format) drives
 - 2400 MB for 3490E Enhanced Cartridge Capacity System Tape (ECCST) drives
 - 30 GB for 3590 drives
 - 100 GB (Media7) or 500 GB (Media5) (physical) for 3592-E05 drives
 - 60 GB (Media7) or 300 GB (Media5) (physical) for 3592-J1A drives
 - 30 GB (Media3) or 60 GB (Media4) (physical) for 3590-H1x drives
- Data compaction allows more data to be stored in less space. Compaction ratios are dependent on such factors as the bit pattern of the data and blocksize of the data. IDRC compaction ratios of 2.5 and higher are not uncommon. The 3590 can achieve compaction ratios of around 3.0.

Tape unit selection

Request A:	By coding:	
3592	UNIT=3590	256/384-track 
3590	UNIT=3590 - 1	128-track 
3590E	UNIT=3490	36-track 
3590/3480 IDRC	UNIT=3480X	18-track 
3490/3480	UNIT=3480	18-track 
3420 Mod 4, 6, 8	UNIT=3400 - 6	9-track 
3420 Mod 3, 5, 7	UNIT=3400 - 4	9-track 

© Copyright IBM Corporation 2011

Figure 12-11. Tape unit selection

ES074.0

Notes:

A single tape cartridge or a single tape reel is one tape volume.

The following UNIT types are used in JCL to request various tape drives (or any esoteric defined in HCD with the corresponding unit type):

- UNIT=3590 requests a 3592 drive.
- UNIT=3590 - 1 requests a 3590 drive.
- UNIT=3490 requests a 3490E drive.
- UNIT=3480X requests a 3490 or 3480 IDRC drive.
- UNIT=3480 requests a 3490 or 3480 drive.
- UNIT=3400 - 9 requests a 3480 in 3420 code compatibility mode.
- UNIT=3400 - 6 requests a dual density 3420, model 4, 6, or 8.
- UNIT=3400 - 5 requests a single density 3420, model 4, 6, or 8.
- UNIT=3400 - 4 requests a dual density 3420, model 3, 5, or 7.
- UNIT=3400 - 3 requests a single density 3420, model 3, 5, or 7.

Tape data class

```

    DATA CLASS DISPLAY          Page 2 of 4

CDS Name . . . : DFSMS.SCDS.ALT
Data Class Name : DB2STRIP

Data Set Name Type . . . : EXTENDED
If Extended . . . . . : REQUIRED
Extended Addressability : NO
Record Access Bias . . : USER
Space Constraint Relief . : NO
Reduce Space Up To (%) :
Dynamic Volume Count . :

{ Compaction . . . . . :
  Spanned / Nonspanned . . . :
  Media Interchange
  Media Type . . . . . :
  Recording Technology . . :
  Performance Scaling . . :


```

Tape attributes

© Copyright IBM Corporation 2011

Figure 12-12. Tape data class

ES074.0

Notes:

A data class provides the tape device selection information for tape data sets.

Data class automatic class selection (ACS) routines direct the tape data set to a specific data class. The data class can be used for SMS and non-SMS tapes. If installations want to take advantage of the performance options, they need to assign non-SMS tapes to an appropriate data class.

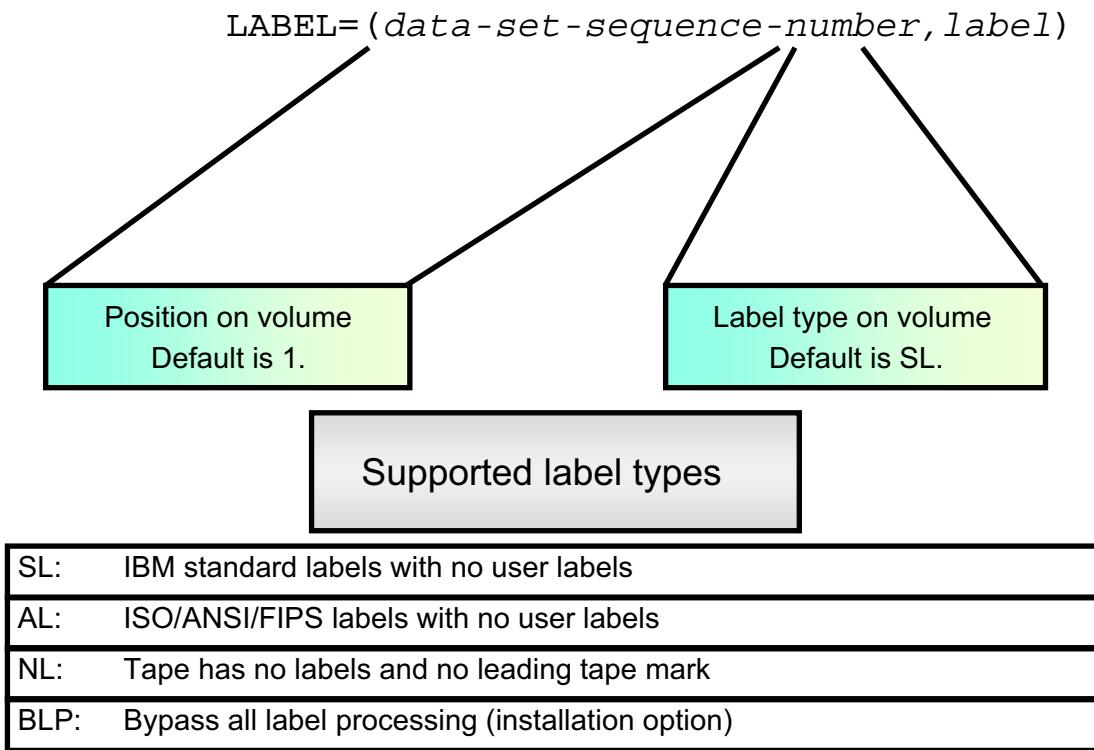
The attributes you can specify are:

- The type of media to use
- Whether the data is to be compacted
- Recording technology (18-track, 36-track, 128-track, or 256-track)

Use Interactive Storage Management Facility (ISMF) panels to define your data classes:

- **Media type:** Specify the tape cartridge type used for data sets associated with this data class.
 - MEDIA1 specifies 3490 CST tapes (standard length).
 - MEDIA2 specifies 3490 ECCST tapes (extended length).
 - MEDIA3 specifies 3590 High Performance Cartridge Tape (HPCT) (standard length).
 - MEDIA4 specifies 3590 Extended High Performance Cartridge Tape (EHPCT) (extended length).
 - MEDIA5: The cartridge is an enterprise tape cartridge 3592.
 - MEDIA6: The cartridge is a 300-GB WORM non-scalable cartridge.
 - MEDIA7: The cartridge is a 60-GB R/W non scalable cartridge.
 - MEDIA8: The cartridge is a 60-GB WORM non scalable cartridge.
- **Recording technology:** Specify the number of tracks on tape cartridges used for data sets associated with this data class.
- Possible values:
 - 18TRACK: The data on the cartridge has 18 recording tracks.
 - 36TRACK: The data on the cartridge has 36 recording tracks.
 - 128TRACK: The data on the cartridge has 128 recording tracks.
 - 256TRACK: The data on the cartridge has 256 recording tracks.
 - 384TRACK: The data on the cartridge has 384 recording tracks.
 - EFMT1: The data on the cartridge is in Enterprise Format 1.

Magnetic tape-label parameter



© Copyright IBM Corporation 2011

Figure 12-13. Magnetic tape-label parameter

ES074.0

Notes:

Some tape volumes have tape labels, and others do not. The **LABEL** parameter specifies:

- Whether a tape has labels
- The tape label format for those volumes that have labels

Additional label types are supported:

- **SUL:** IBM standard labels with user labels
- **AUL:** ISO/ANSI/FIPS labels with user labels
- **LTM:** Tape has no labels but *may* have a leading tape mark
- **NSL:** Tape has nonstandard labels

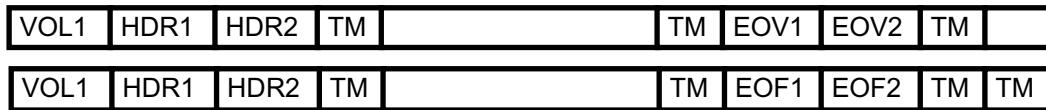
The **LABEL** parameter also has additional subparameters. For example, expiration date (**EXPDT**) and retention period (**RETPD**) are two such parameters.

Tape label format

(IBM standard tape labels)



One data set on one volume



One data set on two volumes



Two data sets on one volume

© Copyright IBM Corporation 2011

Figure 12-14. Tape label format

ES074.0

Notes:

A tape volume can contain one data set, multiple data sets, or it can be part of a set of volumes that contains a single data set. The label format must be the same:

- For all data sets on a single volume.
- For all volumes associated with a multivolume data set.

Five acronyms used in the visual are described here.

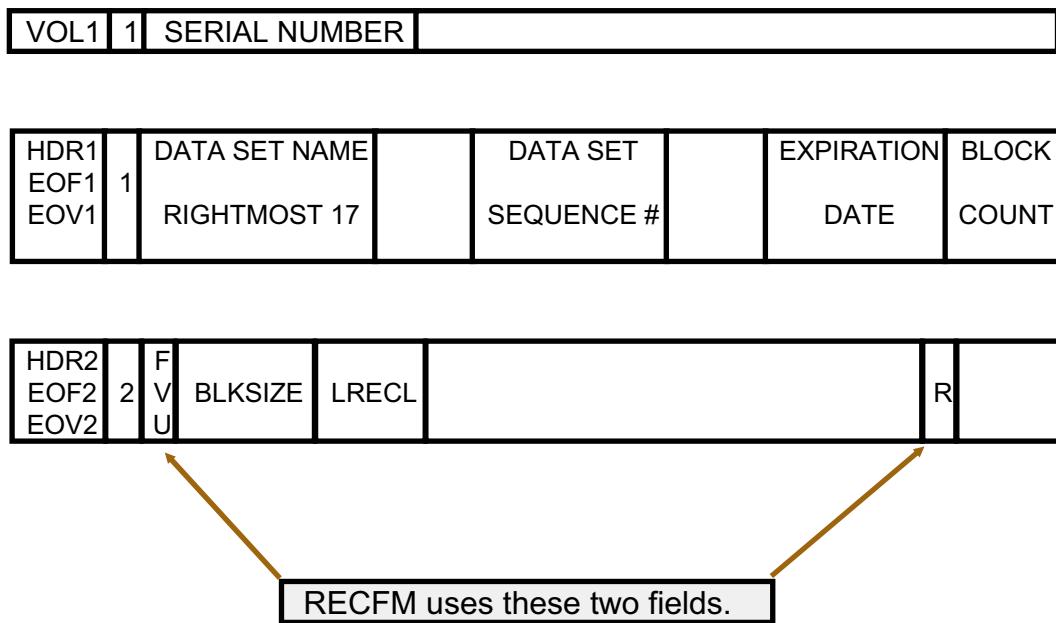
- TM represents a tape mark.
- VOL1 represents a volume label.
- HDRx represents a header label.
- EOFx represents a trailer label.
- EOFx represents an end of volume label.

Only one tape mark follows the trailer label set on all volumes of a multivolume tape data set except the last.

Two label types are valid for DASD: SL and SUL. The default label type is SL.

IBM standard tape labels: Content

(Selected fields)



© Copyright IBM Corporation 2011

Figure 12-15. IBM standard tape labels: Content

ES074.0

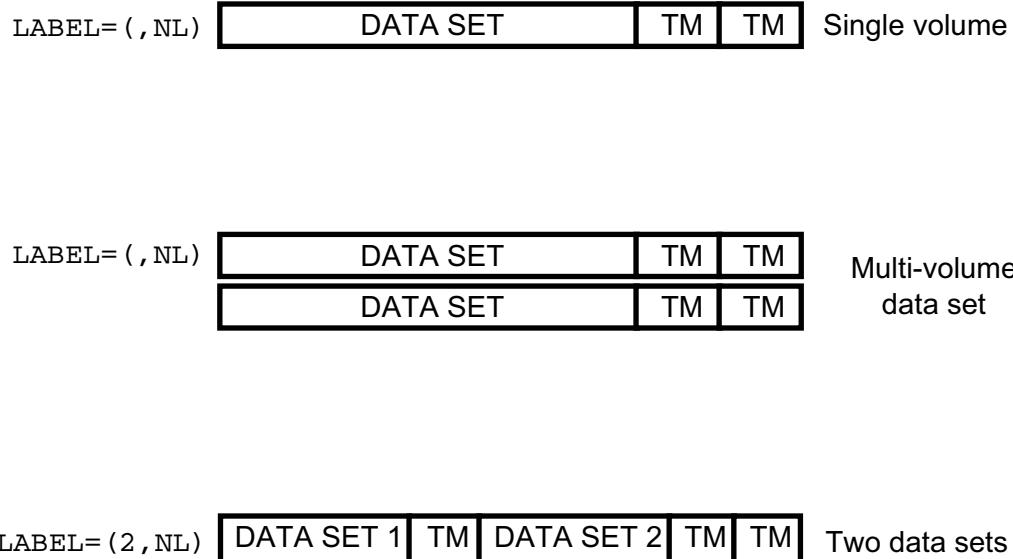
Notes:

Each label record (VOL1, HDR1, HDR2, EOF1, and so on) on an SL tape is 80 bytes long.

Here is some information about the contents of the labels:

- The first four-byte field contains the identifier, VOL1, HDR1, EOF1, and so on.
- The fifth byte is 1 for VOL1, HDR1, EOF1, and EOV1 labels. The fifth byte is 2 for the other labels.
- The serial number is a maximum of six characters, including all numeric and all special character fields.
- Only the rightmost 17 bytes from the data set name are recorded. This includes all periods and other special characters generated by the system. For example, if DSN=MY.FIRST.GDG (+1), the absolute name is DSN=MY.FIRST.GDG.G0001V00. Hence the DSN field contains IRST.GDG.G0001V00.
- RECFM is stored in two one-byte fields. The first field contains F, V, or U. The second field contains B, S, or R. If the second field contains a blank, the data set is unblocked. If RECFM=VBS or RECFM=FBS, the second field contains R.

Unlabeled tapes



© Copyright IBM Corporation 2011

Figure 12-16. Unlabeled tapes

ES074.0

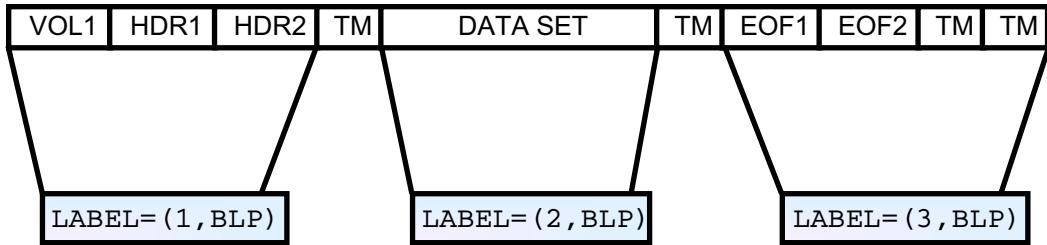
Notes:

Unlabeled tapes contain no tape labels. Thus, tape marks separate tape data sets.

Information such as the volume serial number, data set name (or 17 bytes worth), and DCB attributes are not written to the tape; however, if you request that a tape be cataloged, the volume serial number will still be written to the catalog. Additional information such as DCB attributes might be required on the DD statement to process data on an unlabeled tape.

The system checks the label type. For example, if you specify `LABEL=(_, SL)` on a DD statement and the operator mounts an unlabeled tape, the system rejects it. Also, if you specify `LABEL=(_, NL)` in JCL and the operator mounts a standard label tape, the system rejects it.

Bypass label processing



© Copyright IBM Corporation 2011

Figure 12-17. Bypass label processing

ES074.0

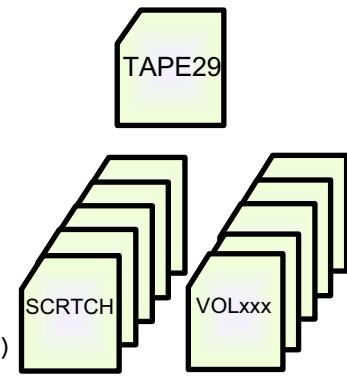
Notes:

`LABEL= (__, BLP)` tells the system to bypass all label processing. Thus `LABEL= (__, BLP)` allows a tape to be processed regardless of whether it has labels or not. For example, if a standard label tape is processed using `LABEL= (1 , BLP)`, the volume label and first two header labels will be processed as data.

Programs can often bypass security checking by processing tapes with `LABEL= (__, BLP)`. For this reason, many shops outlaw the use of BLP. If BLP has been nullified in a shop, the system will change a `LABEL= (__, BLP)` to `LABEL= (__, NL)` and try to process the tape.

Tape allocation: Examples

```
//TAPE1 DD DSN=A.B.C,DISP=(,CATLG),UNIT=3490,VOL=SER=TAPE29
//TAPE2 DD DSN=X.Y.Z,DISP=OLD,UNIT=(3590-1,2)
//TAPE3 DD DSN=X.Y.Z,DISP=OLD,VOL=( , , 7)
//TAPE4 DD DSN=A.B.C,DISP=(MOD,CATLG),
//           VOL=( , , 35),UNIT=( , 2)
//TAPE5 DD DSN=I.J.K,DISP=OLD,UNIT=( , 2,DEFER)
```



© Copyright IBM Corporation 2011

Figure 12-18. Tape allocation: Examples

ES074.0

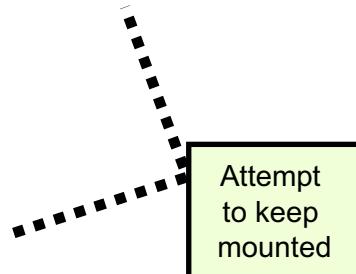
Notes:

The following information pertains to the examples in the visual:

- **TAPE1:** If `DSN=A.B.C` is SMS-managed, `VOL=SER` will be ignored. SMS will select the volumes and allocate one drive. If `DSN=A.B.C` is not SMS-managed, TAPE29 will be mounted on one drive. Up to four additional volumes will be available. A 3490E drive is being requested.
- **TAPE2:** The catalog will be searched to determine which volumes to mount. The volumes will be mounted using two drives in either the SMS or non-SMS managed case.
- **TAPE3:** A catalog search will determine which volumes to mount. One drive will be allocated in the SMS or non-SMS case. Processing will start on the seventh volume in this multivolume data set.
- **TAPE4:** Someone apparently wants to add data to the end of existing data in a multivolume data set. They are asking for up to 35 tape volumes for the step. Two drives will be allocated in either case.
- **TAPE5:** A catalog search will determine the volumes to mount. The first volume will be mounted when the data set is opened. Two drives will be allocated in either case.

Tape mounting: Retain

```
//SAMPLE      JOB     378, SUE, CLASS=T
//STEP1       EXEC    PGM=SOMETHIN
//TAPEOUT     DD      DSN=M.K.E, DISP=(, CATLG), UNIT=TAPE,
//                           VOL=(, RETAIN)
//OTHER        DD      . . .
//STEP2       EXEC    PGM=GOGOGO
//ANOTHER      DD      . . .
//STEP3       EXEC    PGM=HEREWEGO
//TAPEIN      DD      DSN=M.K.E, DISP=(OLD, PASS)
```



- Retain is automatic if `DISP=(, PASS)` is coded.

© Copyright IBM Corporation 2011

Figure 12-19. Tape mounting: Retain

ES074.0

Notes:

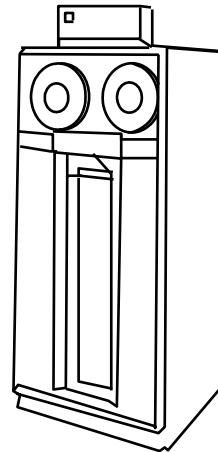
The RETAIN subparameter makes a request to the system to keep a tape volume mounted for use later in the job. If the installation does not have enough drives, the request can be canceled and the tape demounted.

RETAIN is ignored by JES3 installations.

A disposition of PASS accomplishes the same result as RETAIN.

Unit affinity

```
//FRST      DD  DSN=L.G.A,DISP=OLD
//SECOND    DD  DSN=J.F.K,DISP=OLD,UNIT=AFF=FIRST
//THIRD    DD  DSN=E.W.R,DISP=OLD,UNIT=AFF=FIRST
```



© Copyright IBM Corporation 2011

Figure 12-20. Unit affinity

ES074.0

Notes:

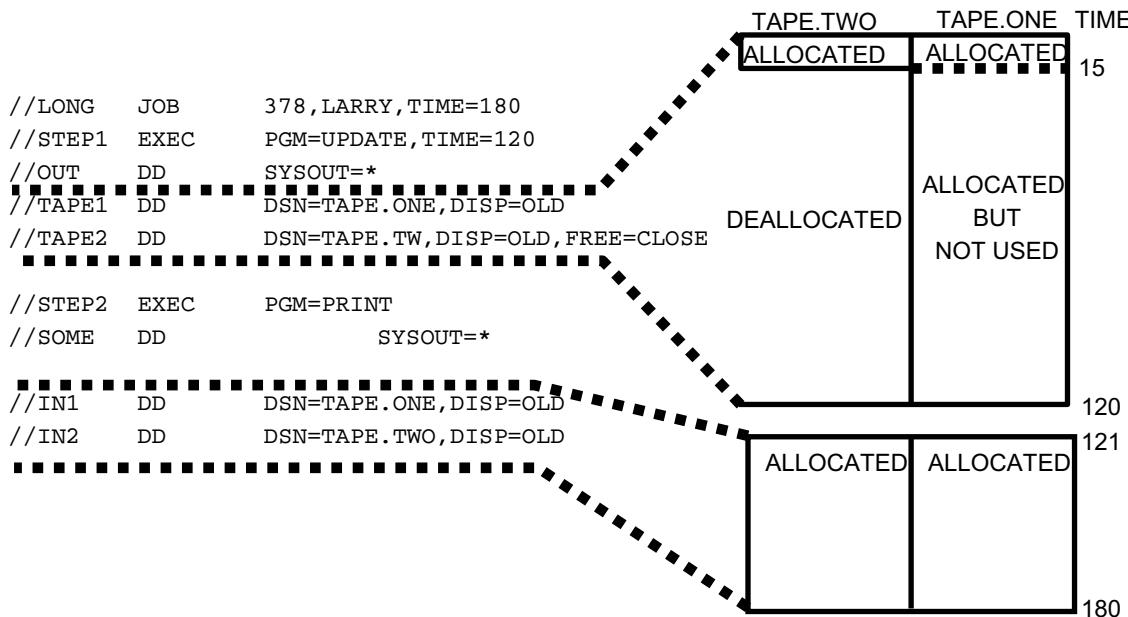
Unit affinity is used to ask the system allocate the same device to *different* data sets on different *removable* volumes.

Unit affinity can be used to reduce the number of devices required to process several data sets on removable volumes.

In the above example, only one device will be allocated to all three data sets.

If unit affinity is specified for SMS-managed DASD, the system ignores it. If unit affinity is specified for SMS-managed tape, the system attempts to honor it. Unit affinity may or may not be honored in JES3 environments.

Tape unit allocation: FREE=CLOSE



© Copyright IBM Corporation 2011

Figure 12-21. Tape unit allocation: FREE=CLOSE

ES074.0

Notes:

FREE=CLOSE can be used for demountable devices such as tapes to request that the device be deallocated when the data set is closed. (FREE=CLOSE can be used in other situations as well. This was discussed earlier in the discussion on print processing.)

A good place to use FREE=CLOSE is in a long-running tape processing step. Assume PGM=UPDATE finishes processing DSN=TAPE.TWO early in step one of job LONG, and assume FREE=CLOSE is coded on the DD statement. The drive will be deallocated when DSN=TAPE.TWO is closed. Therefore, another job can start using the drive before step one of job LONG ends.

Step termination disposition processing occurs when FREE=CLOSE causes deallocation of the device.

New DD parameter FREEVOL was introduced in z/OS V1R13. Specifying FREEVOL=EOV specifies that a tape that is a part of a multivolume data set becomes available at end-of-volume rather than at step end. It allows other jobs to use the tape immediately and it also allows overlapped processing of multivolume tape data sets.

Checkpoint

1. What is the maximum volume count for a data set?
 - a. 8
 - b. 59
 - c. 60

2. How many volumes will be allocated with
UNIT=(3390,2), VOL=(,,4) ?

3. If you want to use a 128-track tape, how would you code the
UNIT parameter?

4. What is the meaning of LABEL=(3,SL) ?

© Copyright IBM Corporation 2011

Figure 12-22. Checkpoint

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Code DD statements to allocate a multivolume DASD data set
- Describe the characteristics of IBM standard tape labels
- Code tape label processing options on the DD statement
- Code DD statements to allocate and locate a multivolume tape data set using single and multiple tape units
- Discuss the types of tape drives and methods of requesting them
- Explain the differences among tape label types
- Code DD statements utilizing the UNIT=AFF=DDNAME parameter to reduce device requirements

© Copyright IBM Corporation 2011

Figure 12-23. Unit summary

ES074.0

Notes:

Unit 13. ABENDs

What this unit is about

This topic will introduce some common ABEND codes and, additionally, the steps in problem determination to resolve them.

What you should be able to do

After completing this unit, you should be able to:

- Define the term ABEND
- Recognize the causes of system ABENDs
- Identify the basic steps in problem determination and recovery
- Use the system codes and messages manuals to aid in performing diagnosis and problem determination

How you will check your progress

- Checkpoint questions
- Machine exercises

References

SA22-7597	<i>z/OS MVS JCL Reference</i>
SA22-7598	<i>z/OS MVS JCL User's Guide</i>
SA22-7631	<i>z/OS MVS System Messages, Vol. 1</i>
SA22-7632	<i>z/OS MVS System Messages, Vol. 2</i>
SA22-7633	<i>z/OS MVS System Messages, Vol. 3</i>
SA22-7634	<i>z/OS MVS System Messages, Vol. 4</i>
SA22-7635	<i>z/OS MVS System Messages, Vol. 5</i>
SA22-7636	<i>z/OS MVS System Messages, Vol. 6</i>
SA22-7637	<i>z/OS MVS System Messages, Vol. 7</i>
SA22-7638	<i>z/OS MVS System Messages, Vol. 8</i>
SA22-7639	<i>z/OS MVS System Messages, Vol. 9</i>
SA22-7640	<i>z/OS MVS System Messages, Vol. 10</i>
SA22-7626	<i>z/OS MVS System Codes</i>

Unit objectives

After completing this unit, you should be able to:

- Define the term ABEND
- Recognize the causes of system ABENDs
- Identify the basic steps in problem determination and recovery
- Use the system codes and messages manuals to aid in performing diagnosis and problem determination

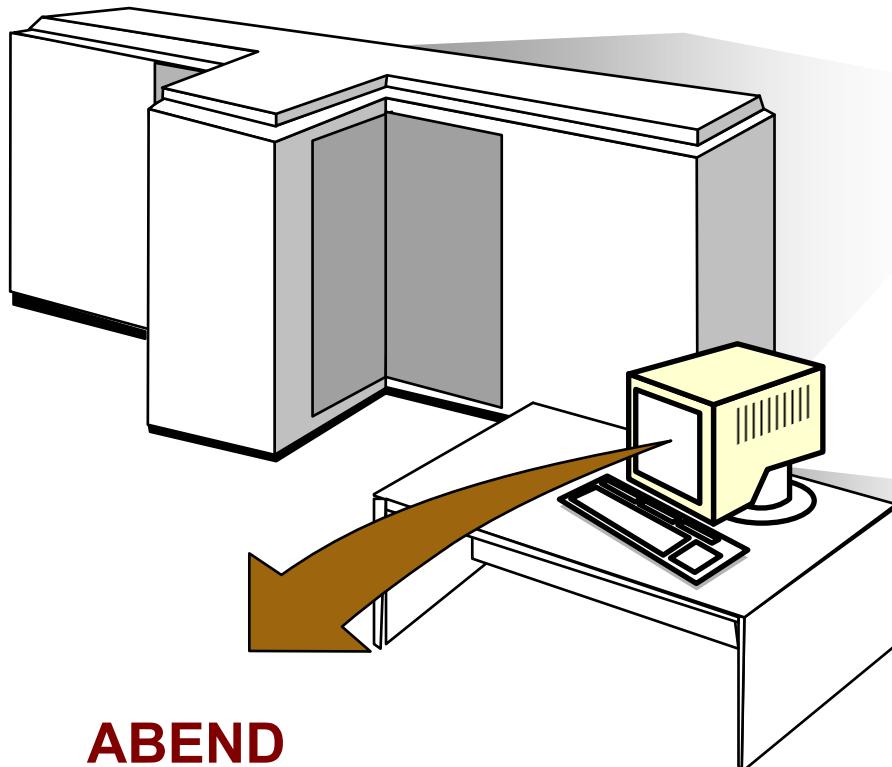
© Copyright IBM Corporation 2011

Figure 13-1. Unit objectives

ES074.0

Notes:

ABENDs



© Copyright IBM Corporation 2011

Figure 13-2. ABENDs

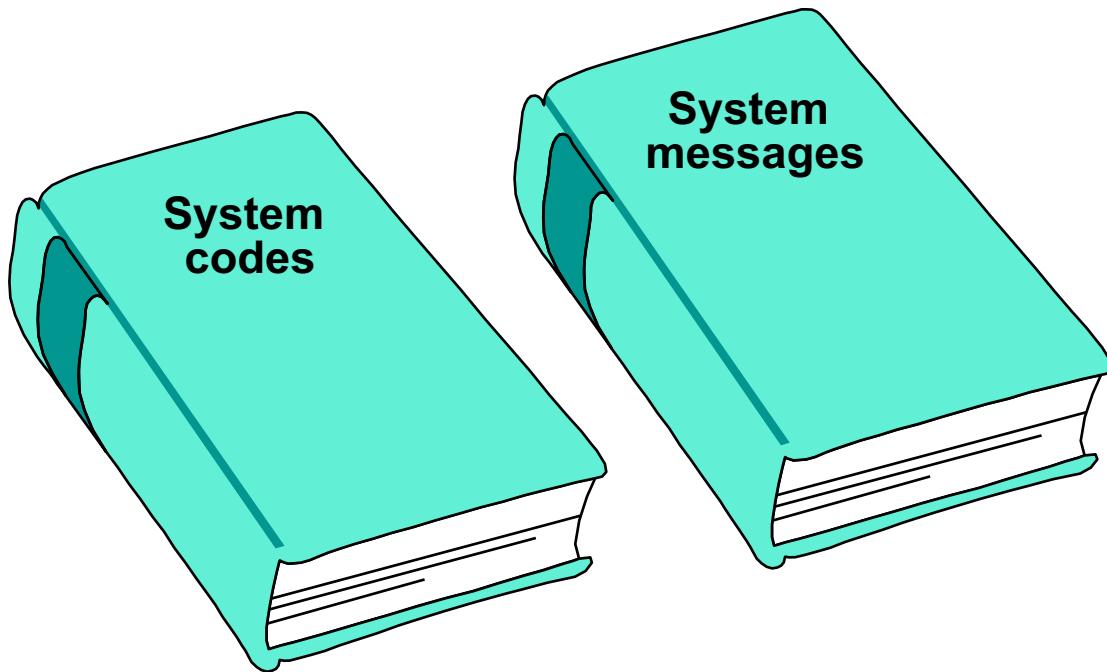
ES074.0

Notes:

A system *ABEND* is an abnormal end or termination of a job or task by the system. The system cannot continue processing and produce valid results.

A system ABEND code is in the form of three hexadecimal characters preceded by the letter S. Example: S222.

ABEND recovery



© Copyright IBM Corporation 2011

Figure 13-3. ABEND recovery

ES074.0

Notes:

System codes include system completion codes (or abend codes) identified by three hexadecimal digits and user completion codes identified by four decimal digits and are usually the result of a system or an application program abnormally ending.

The completion code indicates the reason for the abnormal end.

For recovery of System ABENDs, you will need as reference:

- *z/OS MVS System Codes* and, additionally, you may need:
- *z/OS MVS System Messages Vol 1 to Vol 10*

LookAt on the Internet

The left screenshot shows the 'Messages' search interface. It has three main sections:
 1. A search bar with 'Enter Message ID' and a dropdown for 'Select a country' set to 'z/OS'.
 2. A list of 'Latest Alerts' for 'October 1, 2002' with a link to 'Click here for detail'.
 3. A list of platforms: 'z/OS™ and OS/390™', 'z/OS® and OS/390®', 'z/VM™', 'VSE/ESA™', and 'VSE/ESA®'. Each platform has a radio button next to it, with 'z/VM' and 'VSE/ESA' being selected. Below this is a 'List Books' button.
 At the bottom, there's a 'Feedback' button and a 'Survey' button.

The right screenshot shows a detailed message explanation for 'IEA2001 member-exxx - text'. It includes the message text, an explanation ('Explanation: During system initialization, the system could not use a data set member that was to contain an alternate version of the master scheduler JCL.'), and several error codes listed in a table:

Error Code	Description
IEA2001	member-exxx - text
I/O ERROR DURING RDL	An input/output error occurred when the system used the build list to find the specified module.
I/O ERROR DURING READ	An input/output error occurred when the system was reading the data set that was to contain the alternate version of the master scheduler JCL.
UNEXPECTED END OF FILE	The system found an end-of-file (EOF) before the normal end of processing.
INTERNAL CONVERSION ERROR	An internal error occurred.

<http://www-03.ibm.com/systems/z/os/zos/bkserv/lookat/>

© Copyright IBM Corporation 2011

Figure 13-4. LookAt on the Internet

ES074.0

Notes:

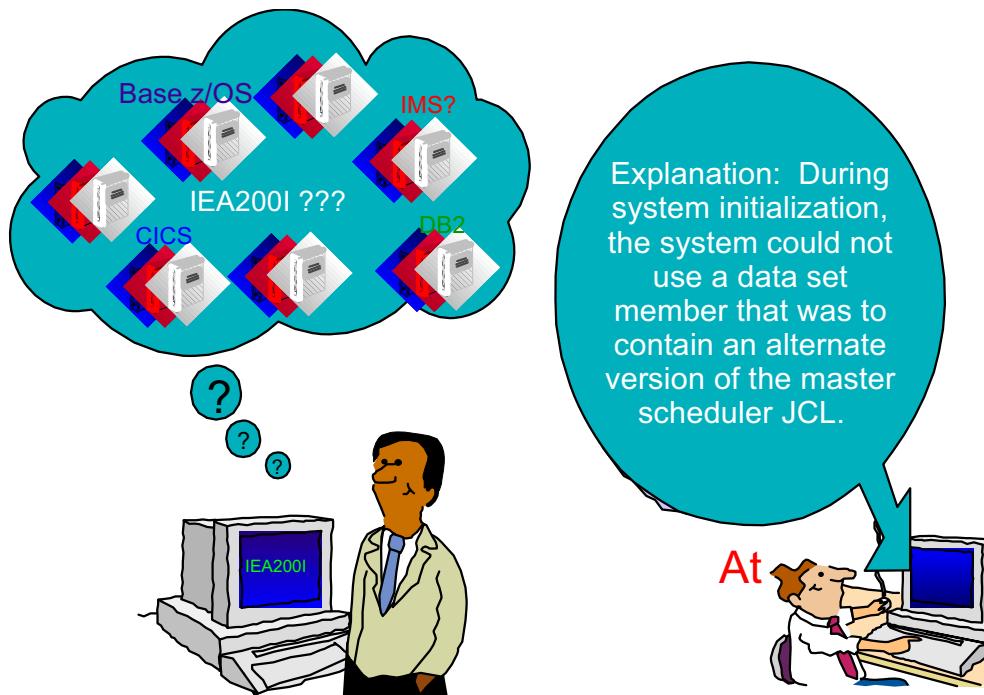
What is LookAt?

- A free message lookUp facility
- Provides fast access to IBM-supplied System z message information
- Currently supports Message Information for OS/390, z/OS, z/VM, and z/VSE
- Indexes over 600,000 IBM messages
- Provides an overall index to message explanations

Why LookAt?

- Message information is not easy to find without experience
- Messages are scattered across large product libraries
- z/OS alone has 420 manuals in the base
- There are over 2000 manuals across IBM applications
- Systems programmers had to “know” product message prefixes
- Programmers wanted easy access to information that is both current and accurate

LookAt for messages and ABEND codes



© Copyright IBM Corporation 2011

Figure 13-5. LookAt for messages and abend codes

ES074.0

Notes:

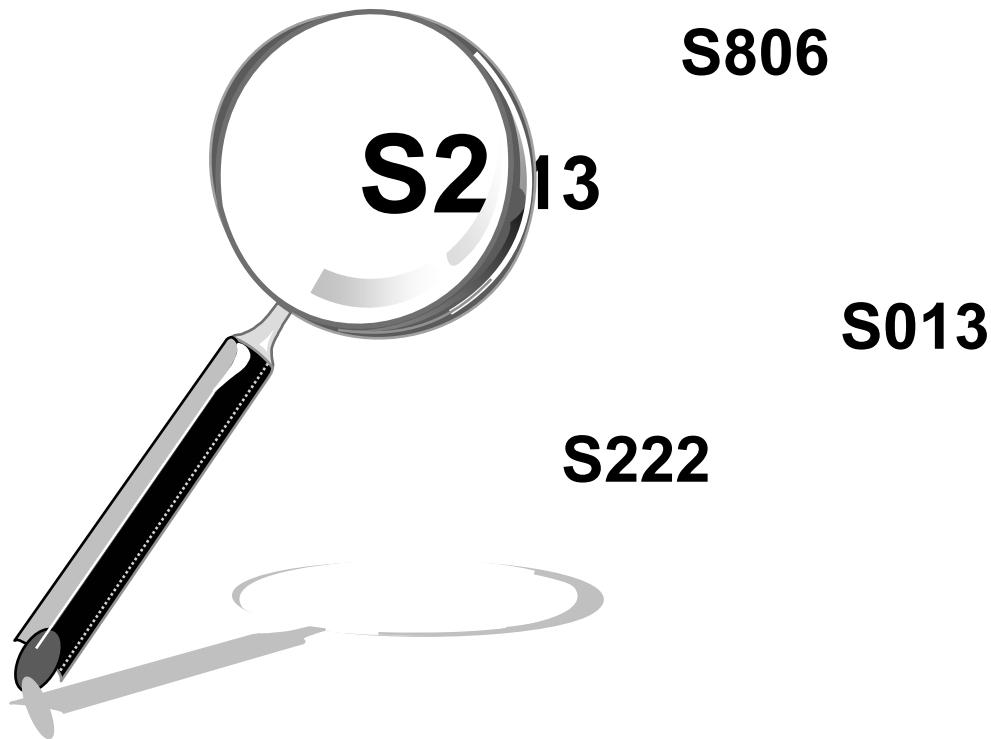
In the **Message ID** field, you enter one of the following:

- The complete message ID for a message to see a specific message.
- Part of a message ID with an asterisk (*) as a wildcard character if you do not know the entire ID or if you want to see a set of related message IDs. The asterisk (*) can represent zero or more character positions. You can place a wildcard character at the beginning, in the middle, or at the end of a message ID.

When you are looking for various codes, if you usually prefix the code with the letters S, ABEND, WSC, or WAIT, LookAt will still attempt to locate the code for you.

- **IEA200I:** LookAt opens directly to this message.
- **IEA21*:** LookAt finds 12 messages that match this pattern in two different books.
- **IEA21*A:** LookAt finds two messages that match this pattern in one book.
- **+012:** LookAt opens directly to the return code.
- **EQA1000I:** LookAt opens directly to the message.
- **0072:** LookAt opens to this ABEND code.
- **9004:** Opens a book to the top of the chapter that contains this code.
- **0123:** Gets a list of books that contain this code.

Common ABEND exercise



© Copyright IBM Corporation 2011

Figure 13-6. Common ABEND exercise

ES074.0

Notes:

Classroom exercise:

The following pages contain:

- A JCL stream
- S222 joblog
- S806-04 joblog
- S013-14 joblog
- S013-18 joblog
- Good run output joblog

JCL STREAM

```

//TSOISBHC JOB 3MACHE3,HENDERS,CLASS=E,MSGCLASS=Q,
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K
//*****STEP1 WILL UNCATLG AND SCRATCH TSOISBH.STEP2.PDS
//*****STEP 1 *****
//** SCRATCH AND UNCATLG SECTION
//*****
//STEP1 EXEC PGM=IEHPRGM
//SYSPRINT DD SYSOUT=*
//DD1 DD VOL=SER=EDPAKX,DISP=OLD,UNIT=SYSDA
//SYSIN DD *
STEP1 SCRATCH DSNAME=TSOISBH.STEP2.PDS,VOL=SYSDA=EDPAK5
STEP1 UNCATLG DSNAME=TSOISBH.STEP2.PDS
//*****STEP 2 *****
//** STEP2 WILL CREATE A NEW PDS CALLED TSOISBH.STEP2.PDS *
//*****
//STEP2 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=ISN.CARDIN.DATA(LAB3),DISP=SHR
//SYSUT2 DD DSN=TSOISBH.STEP2.PDS,DISP=(,CATLG),UNIT=SYSDA,
// VOL=SER=EDPAK5,SPACE=(TRK,(1,1))
//SYSIN DD *
GENERATE MAXNAME=1
MEMBER NAME=NEWDATA
//*****STEP 3 *****
//** STEP3 WILL PRINT THE NEW PDS CALLED TSOISBH.STEP2.PDS **
//*****
//STEP3 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=TSOISBH.STEP2.PDS(DATA),DISP=SHR
//SYSUT2 DD SYSOUT=*,BLKSIZE=133
//SYSIN DD DUMMY

```

© Copyright IBM Corporation 2011

Figure 13-7. JCL STREAM

ES074.0

Notes:

S222 ABEND (1 of 2)

```
J E S 2   J O B   L O G   --   S Y S T E M   M V S 1   --
N O D E   E S S M V S 1

16.08.51 JOB01521 --- FRIDAY, 12 JUN 1998 ---
16.08.51 JOB01521 IRR010I USERID TSOISBH IS ASSIGNED TO THIS JOB.
16.08.52 JOB01521 ICH70001I TSOISBH LAST ACCESS AT 16:05:52 ON FRIDAY, JUNE 12, 1998
16.08.52 JOB01521 #HASP373 TSOISBHC STARTED - INIT 1 - CLASS E - SYS MVS1
16.08.52 JOB01521 IEF403I TSOISBHC - STARTED - TIME=16.08.52
16.08.53 JOB01521 IEF244I TSOISBHC STEP1 - UNABLE TO ALLOCATE 1 UNIT(S)
16.08.53 JOB01521 IEF290E TSOISBHC STEP1 DD1 NEEDS 1 UNIT(S) FOR
VOLUME EDPAKX
OFFLINE NOT ACCESSIBLE
D= 514, 515, 516, 517, 518 D=
D= 519, 51A, 51B, 51C, 51D D=
D= 51E, 51F, 524, 525, 526 D=
D= 527, 534, 535, 536, 537 D=
D= 538, 539, 53A, 53B, 53C D=
D= 53D, 53E, 53F, BC8, BC9 D=
D= BCA, BCB, BCC, BCD, BCE D=
D= BCF, BD0, BD1, BD2, BD3 D=
D= 10B, 120, 122, 22B, 7CB D=
16.08.53 JOB01521 *31 IEF238D TSOISBHC - REPLY DEVICE NAME OR 'CANCEL'.
16.11.23 JOB01521 R 31,CANCEL
16.11.23 JOB01521 IEF251I TSOISBHC JOB CANCELLED
16.11.23 JOB01521 IEF450I TSOISBHC STEP1 - ABEND=S222 U0000 REASON=00000000
16.11.23 JOB01521 #HASP395 TSOISBHC ENDED
----- JES2 JOB STATISTICS -----
12 JUN 1998 JOB EXECUTION DATE
31 CARDS READ
69 SYSOUT PRINT RECORDS
0 SYSOUT PUNCH RECORDS
5 SYSOUT SPOOL KBYTES
2.50 MINUTES EXECUTION TIME
```

© Copyright IBM Corporation 2011

Figure 13-8. S222 ABEND (1 of 2)

ES074.0

Notes:

S222 ABEND (2 of 2)

```

1 //TSOISBHC JOB 3MACHE3BMVS56W,HENDERS,CLASS=E,MSGCLASS=A,           JOB01521
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K
//*****
//*
//*   MAKE SURE YOU CHANGE EDPAK1 AND TSOIOBH IN THE JCL      *
//*
//*****STEP 1 *****
//* SCRATCH AND UNCATLG SECTION
//*****STEP 2 *****
2 //STEP1    EXEC PGM=IEHPRGM
3 //SYSPRINT DD SYSOUT=*
4 //DD1      DD VOL=SER=EDPAKX,DISP=OLD,UNIT=SYSDA
5 //SYSIN DD *
//*****STEP 3 *****
6 //STEP2    EXEC PGM=IEBGENER
7 //SYSPRINT DD SYSOUT=*
8 //SYSUT1   DD DSN=ISN.CARDIN.DATA(LAB3),DISP=SHR
9 //SYSUT2   DD DSN=TSOISBH.STEP2.PDS,DISP=(,CATLG),UNIT=SYSDA,
//          VOL=SER=EDPAK5,SPACE=(TRK,(1,1))
10 //SYSIN   DD *
//*****STEP 4 *****
11 //STEP3   EXEC PGM=IEBGENER
12 //SYSPRINT DD SYSOUT=*
13 //SYSUT1   DD DSN=TSOISBH.STEP2.PDS (DATA),DISP=SHR
14 //SYSUT2   DD SYSOUT=*,BLKSIZE=133
15 //SYSIN   DD DUMMY
ICH70001I TSOISBH LAST ACCESS AT 16:05:52 ON FRIDAY, JUNE 12, 1998
IEF251I TSOISBHC STEP1 - JOB CANCELLED
IEF472I TSOISBHC STEP1 - COMPLETION CODE - SYSTEM=222 USER=0000 REASON=00000000
IEF285I   TSOISBH.TSOISBHC.JOB01521.D0000103.?          SYSOUT
IEF285I   TSOISBH.TSOISBHC.JOB01521.D0000101.?          SYSIN
IEF373I STEP/STEP1 /START 98163.1608
IEF374I STEP/STEP1 /STOP 98163.1611 CPU    0MIN 00.00SEC SRB    0MIN 00.00SEC VIRT    OK SYS    OK EXT    OK SYS    OK
IEF375I JOB/TSOISBHC/START 98163.1608
IEF376I JOB/TSOISBHC/STOP 98163.1611 CPU    0MIN 00.00SEC SRB    0MIN 00.00SEC

```

© Copyright IBM Corporation 2011

Figure 13-9. S222 ABEND (2 of 2)

ES074.0

Notes:

S806-04 ABEND (1 of 3)

```

16.48.39 JOB01522 ---- FRIDAY, 12 JUN 1998 ----
16.44.39 JOB01522 IRR010I USERID TSOISBH IS ASSIGNED TO THIS JOB.
16.44.40 JOB01522 ICH70001I TSOISBH LAST ACCESS AT 16:42:29 ON FRIDAY, JUNE 12, 1998
16.44.40 JOB01522 *HASP373 TSOISBHC STARTED - INIT 1 - CLASS E - SYS MVS1
16.44.40 JOB01522 IEF403I TSOISBHC - STARTED - TIME=16.44.40
16.44.40 JOB01522 CSV003I REQUESTED MODULE IEHPRGM NOT FOUND
16.44.40 JOB01522 CSV028I ABEND806-04 JOBNAM=TSOISBHC STEPNAME=STEP1
16.44.40 JOB01522 IEA995I SYMPTOM DUMP OUTPUT
SYSTEM COMPLETION CODE=806 REASON CODE=00000004
TIME=16.44.39 SEQ=01206 CPU=0000 ASID=0010
PSW AT TIME OF ERROR 070C1000 810AEF9C ILC 2 INTC 0D
NO ACTIVE MODULE FOUND
DATA AT PSW 010AEF96 - A6361810 0A0D186D 58E002FC
GPR 0-3 84806000 84806000 00000000 00FD29D8
GPR 4-7 00000010 007EFC60 007EA450 00F78300
GPR 8-11 010AF986 007EA450 810AEA5A 010AF98A
GPR 12-15 007EF8B8 007EA450 810AEE3A 00000004
END OF SYMPTOM DUMP
16.44.40 JOB01522 IEF450I TSOISBHC STEP1 - ABEND=S806 U0000 REASON=00000004
TIME=16.44.40
16.44.40 JOB01522 IEF404I TSOISBHC - ENDED - TIME=16.44.40
16.44.40 JOB01522 *HASP395 TSOISBHC ENDED
----- JES2 JOB STATISTICS -----
12 JUN 1998 JOB EXECUTION DATE
39 CARDS READ
98 SYSOUT PRINT RECORDS
0 SYSOUT PUNCH RECORDS
7 SYSOUT SPOOL KBYTES
0.00 MINUTES EXECUTION TIME
1 //TSOISBHC JOB 3MACHE3BMVS56W,HENDERS,CLASS=E,MSGCLASS=A,           JOB01522
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K
//*****STEP1 WILL UNCATLG AND SCRATCH TSOISBH.STEP2.PDS   *
//*****STEP 1 ***** STEP 1 *****
//** SCRATCH AND UNCATLG SECTION
//*****STEP1*****STEP2*****STEP3*****STEP4*****STEP5*****STEP6*****

```

© Copyright IBM Corporation 2011

Figure 13-10. S806-04 ABEND (1 of 3)

ES074.0

Notes:

S806-04 ABEND (2 of 3)

```

2 //STEP1    EXEC PGM=IEHPRGM
3 //SYSPRINT DD SYSOUT=*
4 //DD1      DD VOL=SER=EDPAK5,DISP=OLD,UNIT=SYSDA
5 //SYSIN DD *
//***** STEP 2 *****
//* STEP2 WILL CREATE A NEW PDS CALLED TSOISBH.STEP2.PDS *
//*****
6 //STEP2    EXEC PGM=IEBGENER
7 //SYSPRINT DD SYSOUT=*
8 //SYSUT1   DD DSN=ISN.CARDIN.DATA(LAB3),DISP=SHR
9 //SYSUT2   DD DSN=TSOISBH.STEP2.PDS,DISP=(CATLG),UNIT=SYSDA,
//          VOL=SER=EDPAK5,SPACE=(TRK,(1,1))
10 //SYSIN   DD *
//***** STEP 3 *****
//* STEP3 WILL PRINT THE NEW PDS CALLED TSOISBH.STEP2.PDS **
//*****
11 //STEP3   EXEC PGM=IEBGENER
12 //SYSPRINT DD SYSOUT=*
13 //SYSUT1   DD DSN=TSOISBH.STEP2.PDS(DATA),DISP=SHR
14 //SYSUT2   DD SYSOUT=*,BLKSIZE=133
15 //SYSIN   DD DUMMY
ICH70001I TSOISBH LAST ACCESS AT 16:42:29 ON FRIDAY, JUNE 12, 1998
IEF236I ALLOC. FOR TSOISBHC STEP1
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO DD1
IEF237I JES2 ALLOCATED TO SYSIN
CSV003I REQUESTED MODULE IEHPRGM NOT FOUND
CSV028I ABEND806-04 JOBNAME=TSOISBHC STEPNAME=STEP1
IEA995I SYMPTOM DUMP OUTPUT
SYSTEM COMPLETION CODE=806 REASON CODE=00000004
TIME=16.44.39 SEQ=01206 CPU=0000 ASID=0010
PSW AT TIME OF ERROR 070C1000 810AEF9C ILC 2 INTC 0D
NO ACTIVE MODULE FOUND
DATA AT PSW 010AEF96 - A6361810 0A0D186D 58E002FC
GPR 0-3 84806000 84806000 00000000 00FD29D8
GPR 4-7 00000010 007EFC60 007EA450 00F78300

```

© Copyright IBM Corporation 2011

Figure 13-11. S806-04 ABEND (2 of 3)

ES074.0

Notes:

S806-04 ABEND (3 of 3)

```

GPR 8-11 010AF986 007EA450 810AEA5A 010AF98A
GPR 12-15 007EF8B8 007EA450 810AEE3A 00000004
END OF SYMPTOM DUMP
IEF472I TSOISBHC STEP1 - COMPLETION CODE - SYSTEM=806 USER=0000 REASON=00000004
IEF285I TSOISBH.TSOISBHC.JOB01522.D0000103.? SYSOUT
IEF285I SYS98164.T164440.RA000.TSOISBHC.R0001330 KEPT
IEF285I VOL SER NOS= EDPAK5.
IEF285I TSOISBH.TSOISBHC.JOB01522.D0000101.? SYSIN
IEF373I STEP/STEP1 /START 98163.1644
IEF374I STEP/STEP1 /STOP 98163.1644 CPU 0MIN 00.02SEC SRB 0MIN 00.00SEC VIRT 4K SYS 192K EXT 4K SYS 9096K
IEF272I TSOISBHC STEP2 - STEP WAS NOT EXECUTED.
IEF373I STEP/STEP2 /START 98163.1644
IEF374I STEP/STEP2 /STOP 98163.1644 CPU 0MIN 00.00SEC SRB 0MIN 00.00SEC VIRT OK SYS OK EXT OK SYS OK
IEF272I TSOISBHC STEP3 - STEP WAS NOT EXECUTED.
IEF373I STEP/STEP3 /START 98163.1644
IEF374I STEP/STEP3 /STOP 98163.1644 CPU 0MIN 00.00SEC SRB 0MIN 00.00SEC VIRT OK SYS OK EXT OK SYS OK
IEF375I JOB/TSOISBHC/START 98163.1644
IEF376I JOB/TSOISBHC/STOP 98163.1644 CPU 0MIN 00.02SEC SRB 0MIN 0.00SEC

```

© Copyright IBM Corporation 2011

Figure 13-12. S806-04 ABEND (3 of 3)

ES074.0

Notes:

S013-14 ABEND (1 of 4)

```

16.47.28 JOB01523 ---- FRIDAY,    12 JUN 1998 ----
16.47.28 JOB01523 IRR010I USERID TSOISBH IS ASSIGNED TO THIS JOB.
16.47.29 JOB01523 ICH70001I TSOISBH LAST ACCESS AT 16:45:48 ON FRIDAY, JUNE 12, 1998
16.47.29 JOB01523 #HASP373 TSOISBHC STARTED - INIT 1 - CLASS E - SYS MVS1
16.47.29 JOB01523 IEF403I TSOISBHC - STARTED - TIME=16.47.29
16.47.30 JOB01523 IEC141I 013-14,IGG0191B,TSOISBHC,STEP2,SYSUT2,749,EDPAK5,
16.47.30 JOB01523 IEC141I TSOISBH.STEP2.PDS
16.47.30 JOB01523 IEA995I SYMPTOM DUMP OUTPUT
SYSTEM COMPLETION CODE=013
TIME=16.47.29 SEQ=01220 CPU=0000 ASID=0010
PSW AT TIME OF ERROR 075C1000 00DD99DC ILC 2 INTC 0D
NO ACTIVE MODULE FOUND
DATA AT PSW 00DD99D6 - 4100395E 0A0D4DE0 39025820
GPR 0-3 00DD9B98 A0013000 00008648 00DD923A
GPR 4-7 007EA8C0 007EAC44 007EABF4 007EAC44
GPR 8-11 007EAC14 00FD29D8 18FC7CE8 007EA7EC
GPR 12-15 00FBAA6C8 00000000 00DD9390 00000014
END OF SYMPTOM DUMP
16.47.30 JOB01523 IEF450I TSOISBHC STEP2 - ABEND=S013 U0000 REASON=00000000
TIME=16.47.30
16.47.30 JOB01523 IEF404I TSOISBHC - ENDED - TIME=16.47.30
16.47.30 JOB01523 #HASP395 TSOISBHC ENDED
----- JES2 JOB STATISTICS -----
12 JUN 1998 JOB EXECUTION DATE
      39 CARDS READ
     122 SYSOUT PRINT RECORDS
       0 SYSOUT PUNCH RECORDS
       8 SYSOUT SPOOL KBYTES
  0.01 MINUTES EXECUTION TIME
1 //TSOISBHC JOB 3MACHE3BMVS56W,HENDERS,CLASS=E,MSGCLASS=A,           JOB01523
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K
//*****STEP1 WILL UNCATLG AND SCRATCH TSOISBH.STEP2.PDS   *
//*
//*
//*****STEP 1 *****
//* SCRATCH AND UNCATLG SECTION
//*****
```

© Copyright IBM Corporation 2011

Figure 13-13. S013-14 ABEND (1 of 4)

ES074.0

Notes:

S013-14 ABEND (2 of 4)

```

2 //STEP1 EXEC PGM=IEHPROGM
3 //SYSPRINT DD SYSOUT=*
4 //DD1 DD VOL=SER=EDPAK5,DISP=OLD,UNIT=SYSDA
5 //SYSIN DD *
//***** STEP 2 *****
//*
//** STEP2 WILL CREATE A NEW PDS CALLED TSOISBH.STEP2.PDS *
//*
//***** STEP2 *****
6 //STEP2 EXEC PGM=IEBGENER
7 //SYSPRINT DD SYSOUT=*
8 //SYSUT1 DD DSN=ISN.CARDIN.DATA(LAB3),DISP=SHR
9 //SYSUT2 DD DSN=TSOISBH.STEP2.PDS,DISP=(,CATLG),UNIT=SYSDA,
//      VOL=SER=EDPAK5,SPACE=(TRK,(1,1))
10 //SYSIN DD *
//***** STEP 3 *****
//*
//** STEP3 WILL PRINT THE NEW PDS CALLED TSOISBH.STEP2.PDS **
//*
//***** STEP3 *****
11 //STEP3 EXEC PGM=IEBGENER
12 //SYSPRINT DD SYSOUT=*
13 //SYSUT1 DD DSN=TSOISBH.STEP2.PDS(DATA),DISP=SHR
14 //SYSUT2 DD SYSOUT=*,BLKSIZE=133
15 //SYSIN DD DUMMY
ICH700011 TSOISBH LAST ACCESS AT 16:45:48 ON FRIDAY, JUNE 12, 1998
IEF236I ALLOC. FOR TSOISBHC STEP1
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO DD1
IEF237I JES2 ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP1 - STEP WAS EXECUTED - COND CODE 0000
IEF285I TSOISBH.TSOISBHC.JOB01523.D0000103.?          SYSOUT
IEF285I SY98164.T164729.RA000.TSOISBHC.R0001332        KEPT
IEF285I VOL SER NOS= EDPAK5.
IEF285I TSOISBH.TSOISBHC.JOB01523.D0000101.?          SYSIN
IEF373I STEP/STEP1 /START 98163.1647
IEF374I STEP/STEP1 /STOP 98163.1647 CPU    OMIN 00.03SEC SRB    OMIN 00.00SEC VIRT    48K SYS   244K EXT    4K SYS   9124K

```

© Copyright IBM Corporation 2011

Figure 13-14. S013-14 ABEND (2 of 4)

ES074.0

Notes:

S013-14 ABEND (3 of 4)

```

IEF236I ALLOC. FOR TSOISBHC STEP2
IEF237I JES2 ALLOCATED TO SYSPRINT
IGD103I SMS ALLOCATED TO DDNAME SYSUT1
IGD100I 749 ALLOCATED TO DDNAME SYSUT2 DATACLAS ( )
IEF237I JES2 ALLOCATED TO SYSIN
IEC141I 013-14, IGG0191B,TSOISBHC,STEP2,SYSUT2,749,EDPAK5,
IEC141I TSOISBH.STEP2.PDS
IEA995I SYMPTON DUMP OUTPUT
SYSTEM COMPLETION CODE=013
TIME=16.47.29 SEQ=01220 CPU=0000 ASID=0010
PSW AT TIME OF ERROR 075C1000 00DD99DC ILC 2 INTC 0D
NO ACTIVE MODULE FOUND
DATA AT PSW 00DD99D6 - 4100395E 0A0D4DE0 39025820
GPR 0-3 00DD9B98 A0013000 0000B648 00DD923A
GPR 4-7 007EA8C0 007EAC44 007EABF4 007EAC44
GPR 8-11 007EAC14 00FD29D8 18FC7CE8 007EA7EC
GPR 12-15 00FBA6C8 00000000 00DD9390 00000014
END OF SYMPTON DUMP
IEF472I TSOISBHC STEP2 - COMPLETION CODE - SYSTEM=013 USER=0000 REASON=00000000
IEF285I TSOISBH.TSOISBHC.JOB01523.D0000104.?
SYSOUT
IGD104I ISN.CARDIN.DATA RETAINED, DDNAME=SYSUT1
IEF285I TSOISBH.STEP2.PDS CATALOGED
IEF285I VOL SER NOS= EDPAK5.
IEF285I TSOISBH.TSOISBHC.JOB01523.D0000102.?
SYSIN
IEF373I STEP/STEP2 /START 98163.1647
IEF374I STEP/STEP2 /STOP 98163.1647 CPU 0MIN 00.04SEC SRB 0MIN 00.00SEC VIRT 48K SYS 236K EXT 4K SYS 9112K
IEF272I TSOISBHC STEP3 - STEP WAS NOT EXECUTED.
IEF373I STEP/STEP3 /START 98163.1647
IEF374I STEP/STEP3 /STOP 98163.1647 CPU 0MIN 00.00SEC SRB 0MIN 00.00SEC VIRT OK SYS OK EXT OK SYS OK
IEF375I JOB/TSOISBHC/START 98163.1647
IEF376I JOB/TSOISBHC/STOP 98163.1647 CPU 0MIN 00.07SEC SRB 0MIN 00.00SEC
SYSTEM SUPPORT UTILITIES ---- IEHPROGM
PAGE 0001

STEP1 SCRATCH DSNAME=TSOISBH.STEP2.PDS,VOL=SYSDA=EDPAK5 00043301
NORMAL END OF TASK RETURNED FROM SCRATCH

STEP1 UNCATLG DSNAME=TSOISBH.STEP2.PDS 00043601
IEH210I YOUR REQUEST CANNOT BE SERVICED.....
IT IS INVALID --OR-- IT IS NOT PROPER WITHIN YOUR PRESENT CATALOG STRUCTURE ... UNUSUAL END

```

© Copyright IBM Corporation 2011

Figure 13-15. S013-14 ABEND (3 of 4)

ES074.0

Notes:

S013-14 ABEND (4 of 4)

```
UTILITY END
DATA SET UTILITY - GENERATE
GENERATE MAXNAME=1
MEMBER NAME=NEWDATA
IEB352I WARNING : OUTPUT RECFM/LRECL COPIED FROM INPUT
                                         PAGE 0001
                                         00045900
                                         00046001
```

© Copyright IBM Corporation 2011

Figure 13-16. S013-14 ABEND (4 of 4)

ES074.0

Notes:

S013-18 ABEND (1 of 4)

```

16.52.49 JOB01524 ---- FRIDAY,    12 JUN 1998 ----
16.52.49 JOB01524 IRR010I USERID TSOISBH IS ASSIGNED TO THIS JOB.
16.52.49 JOB01524 ICH70001I TSOISBH LAST ACCESS AT 16:50:23 ON FRIDAY, JUNE 12, 1998
16.52.49 JOB01524 #HASP373 TSOISBHC STARTED - INIT 1 - CLASS E - SYS MVS1
16.52.49 JOB01524 IEF403I TSOISBHC - STARTED - TIME=16.52.49
16.52.51 JOB01524 IEC141I 013-18,IGG0191B,TSOISBHC,STEP3,SYSUT1,749,EDPAK5,
16.52.51 JOB01524 IEC141I TSOISBH.STEP2.PDS
16.52.51 JOB01524 IEA995I SYMPTOM DUMP OUTPUT
SYSTEM COMPLETION CODE=013
TIME=16.52.50 SEQ=01238 CPU=0000 ASID=0010
PSW AT TIME OF ERROR 075C1000 00DD99DC ILC 2 INTC 0D
NO ACTIVE MODULE FOUND
DATA AT PSW 00DD99D6 - 4100395E 0A0D4DE0 39025820
GPR 0-3 00DD9B98 A0013000 0000B5E8 00DD923A
GPR 4-7 007EAD68 007EA34C 007EA2FC 007EA34C
GPR 8-11 007EA31C 00FD29D8 58FC7CE8 007EA0B4
GPR 12-15 00000008 00000000 00DD9390 00000018
END OF SYMPTOM DUMP
16.52.51 JOB01524 IEF450I TSOISBHC STEP3 - ABEND=S013 U0000 REASON=00000000
TIME=16.52.51
16.52.51 JOB01524 IEF404I TSOISBHC - ENDED - TIME=16.52.51
16.52.51 JOB01524 #HASP395 TSOISBHC ENDED
----- JES2 JOB STATISTICS -----
12 JUN 1998 JOB EXECUTION DATE
 39 CARDS READ
 133 SYSOUT PRINT RECORDS
   0 SYSOUT PUNCH RECORDS
   9 SYSOUT SPOOL KBYTES
 0.02 MINUTES EXECUTION TIME
1 //TSOISBHC JOB 3MACHE3BMVS56W,HENDERS,CLASS=E,MSGCLASS=A,          JOB01524
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K
//*****STEP1 WILL UNCATLG AND SCRATCH TSOISBH.STEP2.PDS      *
//** STEP1 WILL UNCATLG AND SCRATCH TSOISBH.STEP2.PDS      *

```

© Copyright IBM Corporation 2011

Figure 13-17. S013-18 ABEND (1 of 4)

ES074.0

Notes:

S013-18 ABEND (2 of 4)

```

//***** STEP 1 *****
//* SCRATCH AND UNCATLG SECTION
//*****
2 //STEP1 EXEC PGM=IEHPROGM
3 //SYSPRINT DD SYSOUT=*
4 //DD1 DD VOL=SER=EDPAK5,DISP=OLD,UNIT=SYSDA
5 //SYSIN DD *
//***** STEP 2 *****
//*
//* STEP2 WILL CREATE A NEW PDS CALLED TSOISBH.STEP2.PDS *
//*
//*****
6 //STEP2 EXEC PGM=IEBGENER
7 //SYSPRINT DD SYSOUT=*
8 //SYSUT1 DD DSN=ISN.CARDIN,DATA(LAB3),DISP=SHR
9 //SYSUT2 DD DSN=TSOISBH.STEP2.PDS,DISP=(,CATLG),UNIT=SYSDA,
// VOL=SER=EDPAK5,SPACE=(TRK,(1,1,1))
10 //SYSIN DD *
//***** STEP 3 *****
//*
//* STEP3 WILL PRINT THE NEW PDS CALLED TSOISBH.STEP2.PDS **
//*
//*****
11 //STEP3 EXEC PGM=IEBGENER
12 //SYSPRINT DD SYSOUT=*
13 //SYSUT1 DD DSN=TSOISBH.STEP2.PDS(DATA),DISP=SHR
14 //SYSUT2 DD SYSOUT=*,BLKSIZE=133
15 //SYSIN DD DUMMY

ICH70001I TSOISBH LAST ACCESS AT 16:50:23 ON FRIDAY, JUNE 12, 1998
IEF236I ALLOC FOR TSOISBHC STEP1
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO DD1
IEF237I JES2 ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP1 - STEP WAS EXECUTED - COND CODE 0000
IEF285I TSOISBH.TSOISBHC.JOB01524.D0000103.?      SYSOUT
IEF285I SY98164.T165249.RA000.TSOISBHC.R0001335      KEPT
IEF285I VOL SER NOS= EDPAK5.
IEF285I TSOISBH.TSOISBHC.JOB01524.D0000101.?      SYSIN
IEF373I STEP/STEP1 /START 98163.1652
IEF374I STEP/STEP1 /STOP 98163.1652 CPU    0MIN 00.03SEC SRB    0MIN 00.00SEC VIRT    48K SYS   244K EXT    4K SYS   9124K

```

© Copyright IBM Corporation 2011

Figure 13-18. S013-18 ABEND (2 of 4)

ES074.0

Notes:

S013-18 ABEND (3 of 4)

```
IEF236I ALLOC. FOR TSOISBHC STEP2
IEF237I JES2 ALLOCATED TO SYSPRINT
IGD103I SMS ALLOCATED TO DDNAME SYSUT1
IGD100I 749 ALLOCATED TO DDNAME SYSUT2 DATACLAS ( )
IEF237I JES2 ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP2 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   TSOISBHC.JOB01524.D0000104.?
IEF104I ISN.CARDIN.DATA          RETAINED, DDNAME=SYSUT1
IEF285I   TSOISBHC.STEP2.PDS    CATALOGED
IEF285I VOL SER NOS= EDPAK5.
IEF285I   TSOISBHC.JOB01524.D0000102.?
IEF373I STEP/STEP2 /START 98163.1652          SYSIN
IEF374I STEP/STEP2 /STOP  98163.1652 CPU    0MIN 00.03SEC SRB   0MIN 00.00SEC VIRT   96K SYS   236K EXT   4K SYS   9112K
IEF236I ALLOC. FOR TSOISBHC STEP3
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO SYSUT1
IEF237I JES2 ALLOCATED TO SYSUT2
IEF237I DMY ALLOCATED TO SYSIN
IEC141I 013-18,IGG0191B,TSOISBHC,STEP3,SYSUT1,749,EDPAK5,
IEC141I TSOISBHC.STEP2.PDS
IEA995I SYMPTOM DUMP OUTPUT
SYSTEM COMPLETION CODE=013
TIME=16.52.50 SEQ=01238 CPU=0000 ASID=0010
PSW AT TIME OF ERROR 075C1000 00DD99DC ILC 2 INTC 0D
NO ACTIVE MODULE FOUND
DATA AT PSW 00DD99D6 - 4100395E 0A0D4DE0 39025820
GPR 0-3 00DD9B98 A0013000 0000B5E8 00DD923A
GPR 4-7 007EA68 007EA34C 007EA2FC 007EA34C
GPR 8-11 007EA31C 00FD29D8 58FC7CE8 007EA0B4
GPR 12-15 00000008 00000000 00DD9390 00000018
END OF SYMPTOM DUMP
IEF472I TSOISBHC STEP3 - COMPLETION CODE - SYSTEM=013 USER=0000 REASON=00000000
IEF285I   TSOISBHC.JOB01524.D0000105.?
IEF285I   TSOISBHC.STEP2.PDS    KEPT
IEF285I VOL SER NOS= EDPAK5.
IEF285I   TSOISBHC.JOB01524.D0000106.?
SYSOUT
```

© Copyright IBM Corporation 2011

Figure 13-19. S013-18 ABEND (3 of 4)

ES074.0

Notes:

S013-18 ABEND (4 of 4)

```
IEF373I STEP/STEP3 /START 98163.1652
IEF374I STEP/STEP3 /STOP 98163.1652 CPU    0MIN 00.03SEC SRB    0MIN 00.00SEC VIRT    48K SYS   200K EXT    4K SYS   9112K
IEF375I JOB/TSOISBHC/START 98163.1652
IEF376I JOB/TSOISBHC/STOP 98163.1652 CPU    0MIN 00.09SEC SRB    0MIN 00.00SEC
SYSTEM SUPPORT UTILITIES ---- IEHPROGM
                                                PAGE 0001

STEP1 SCRATCH DSNAME=TSOISBH.STEP2.PDS,VOL=SYSDA=EDPAKS      00043301
NORMAL END OF TASK RETURNED FROM SCRATCH

STEP1 UNCATLG DSNAME=TSOISBH.STEP2.PDS                      00043601
NORMAL END OF TASK RETURNED FROM UNCATLG

UTILITY END
DATA SET UTILITY - GENERATE
GENERATE MAXNAME=1
MEMBER NAME=NEWDATA                                         PAGE 0001
00045900
00046001

IEB352I WARNING : OUTPUT RECFM/LRECL COPIED FROM INPUT

PROCESSING ENDED AT EOD
DATA SET UTILITY - GENERATE                                 PAGE 0001
```

© Copyright IBM Corporation 2011

Figure 13-20. S013-18 ABEND (4 of 4)

ES074.0

Notes:

Good run (1 of 4)

```

16.53.29 JOB01525 ---- FRIDAY, 12 JUN 1998 ----
16.53.29 JOB01525 IRR010I USERID TSOISBH IS ASSIGNED TO THIS JOB.
16.53.30 JOB01525 ICH70001I TSOISBH LAST ACCESS AT 16:52:49 ON FRIDAY, JUNE 12, 1998
16.53.30 JOB01525 *HASP373 TSOISBHC STARTED - INIT 1 - CLASS E - SYS MVS1
16.53.30 JOB01525 IEF403I TSOISBHC - STARTED - TIME=16.53.30
16.53.31 JOB01525 IEF404I TSOISBHC - ENDED - TIME=16.53.31
16.53.31 JOB01525 *HASP395 TSOISBHC ENDED
----- JES2 JOB STATISTICS -----
12 JUN 1998 JOB EXECUTION DATE
    39 CARDS READ
    143 SYSOUT PRINT RECORDS
        0 SYSOUT PUNCH RECORDS
        10 SYSOUT SPOOL KBYTES
    0.02 MINUTES EXECUTION TIME
1 //TSOISBHC JOB 3MACHE3BMVS56W,HENDERS,CLASS=E,MSGCLASS=A,           JOB01525
// NOTIFY=TSOISBH,MSGLEVEL=(1,1),REGION=512K
//*****
//*
//**      STEP1 WILL UNCATLG AND SCRATCH TSOISBH.STEP2.PDS      *
//*
//***** STEP 1 *****
//** SCRATCH AND UNCATLG SECTION
//***** *****
2 //STEP1 EXEC PGM=IEHPROGM
3 //SYSPRINT DD SYSOUT=*
4 //DD1 DD VOL=SER=EDPAK5,DISP=OLD,UNIT=SYSDA
5 //SYSIN DD *
//***** STEP 2 *****
//*
//**      STEP2 WILL CREATE A NEW PDS CALLED TSOISBH.STEP2.PDS *
//*
//***** *****

```

© Copyright IBM Corporation 2011

Figure 13-21. Good run (1 of 4)

ES074.0

Notes:

Good run (2 of 4)

```

6 //STEP2      EXEC PGM=IEBGENER
7 //SYSPRINT DD SYSOUT=*
8 //SYSUT1   DD DSN=ISN.CARDIN,DATA(LAB3),DISP=SHR
9 //SYSUT2   DD DSN=TSOISBH.STEP2.PDS,DISP=(,CATLG),UNIT=SYSDA,
//           VOL=SER=EDPAK5,SPACE=(TRK,(1,1,1))
10 //SYSIN    DD *
//***** STEP 3 *****
//*
//** STEP3 WILL PRINT THE NEW PDS CALLED TSOISBH.STEP2.PDS **
//*
//*****
11 //STEP3      EXEC PGM=IEBGENER
12 //SYSPRINT DD SYSOUT=*
13 //SYSUT1   DD DSN=TSOISBH.STEP2.PDS(NEWDATA),DISP=SHR
14 //SYSUT2   DD SYSOUT=*,BLKSIZE=133
15 //SYSIN    DD DUMMY
ICH70001I TSOISBH LAST ACCESS AT 16:52:49 ON FRIDAY, JUNE 12, 1998
IEF236I ALLOC. FOR TSOISBHC STEP1
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO DD1
IEF237I JES2 ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP1 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   TSOISBH.TSOISBHC.JOB01525.D0000103.?          SYSOUT
IEF285I   SYS98164.T165330.RA000.TSOISBHC.R0001336        KEPT
IEF285I   VOL SER NOS= EDPAK5.
IEF285I   TSOISBH.TSOISBHC.JOB01525.D0000101.?          SYSIN
IEF373I STEP/STEP1 /START 98163.1653
IEF374I STEP/STEP1 /STOP 98163.1653 CPU     0MIN 00.03SEC SRB    0MIN 00.00SEC VIRT    48K SYS   244K EXT    4K SYS   9124K
IEF236I ALLOC. FOR TSOISBHC STEP2
IEF237I JES2 ALLOCATED TO SYSIN
IGD103I SMS ALLOCATED TO DDNAME SYSUT1
IGD100I 749 ALLOCATED TO DDNAME SYSUT2 DATACLAS (      )
IEF237I JES2 ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP2 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   TSOISBH.TSOISBHC.JOB01525.D0000104.?          SYSOUT
IGD104I ISN.CARDIN,DATA                           RETAINED,  DDNAME=SYSUT1
IEF285I   TSOISBH.STEP2.PDS                         CATALOGED
IEF285I   VOL SER NOS= EDPAK5.
IEF285I   TSOISBH.TSOISBHC.JOB01525.D0000102.?          SYSIN

```

© Copyright IBM Corporation 2011

Figure 13-22. Good run (2 of 4)

ES074.0

Notes:

Good run (3 of 4)

```

IEF373I STEP/STEP2 /START 98163.1653
IEF374I STEP/STEP2 /STOP 98163.1653 CPU    0MIN 00.04SEC SRB   0MIN 00.00SEC VIRT   96K SYS  236K EXT   4K SYS  9112K
IEF236I ALLOC. FOR TSOISBHC STEP3
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF237I 749 ALLOCATED TO SYSUT1
IEF237I JES2 ALLOCATED TO SYSUT2
IEF237I DMY ALLOCATED TO SYSIN
IEF142I TSOISBHC STEP3 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   TSOISBHC.TSOISBHC.JOB01525.D0000105.?          SYSOUT
IEF285I   TSOISBHC.STEP2.PDS                                KEPT
IEF285I   VOL SER NOS= EDPAK5.
IEF285I   TSOISBHC.TSOISBHC.JOB01525.D0000106.?          SYSOUT
IEF373I STEP/STEP3 /START 98163.1653
IEF374I STEP/STEP3 /STOP 98163.1653 CPU    0MIN 00.02SEC SRB   0MIN 00.00SEC VIRT   76K SYS  196K EXT   4K SYS  9100K
IEF375I JOB/TSOISBHC/START 98163.1653
IEF376I JOB/TSOISBHC/STOP 98163.1653 CPU    0MIN 00.09SEC SRB   0MIN 00.00SEC
SYSTEM SUPPORT UTILITIES ---- IEHPROGM
                                                PAGE 0001

STEP1 SCRATCH DSNAME=TSOISBHC.STEP2.PDS,VOL=SYSDA=EDPAK5      00043301
NORMAL END OF TASK RETURNED FROM SCRATCH

STEP1 UNCATLG DSNAME=TSOISBHC.STEP2.PDS                      00043601
NORMAL END OF TASK RETURNED FROM UNCATLG

UTILITY END
DATA SET UTILITY - GENERATE
GENERATE MAXNAME=1                                         PAGE 0001
MEMBER NAME=NEWDATA                                         00045900
00046001
IEB352I WARNING : OUTPUT RECFM/LRECL COPIED FROM INPUT

PROCESSING ENDED AT EOD
DATA SET UTILITY - GENERATE
                                         PAGE 0001

PROCESSING ENDED AT EOD

```

© Copyright IBM Corporation 2011

Figure 13-23. Good run (3 of 4)

ES074.0

Notes:

Good run (4 of 4)

```
*****
*****          ! ! !   C O N G R A T U L A T I O N S   ! ! !
*****
**          THIS JOB EXECUTED SUCCESSFULLY.  YOU ARE FINISHED!
*****
**          IBM EDUCATION AND TRAINING      z/OS JCL AND UTILITIES
*****
*****
```

© Copyright IBM Corporation 2011

Figure 13-24. Good run (4 of 4)

ES074.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Define the term ABEND
- Recognize the causes of system ABENDs
- Identify the basic steps in problem determination and recovery
- Use the system codes and messages manuals to aid in performing diagnosis and problem determination

© Copyright IBM Corporation 2011

Figure 13-25. Unit summary

ES074.0

Notes:

Appendix A. Checkpoint solutions

Unit 1

Checkpoint solutions (1 of 2)

1. Which of the following are valid Job Entry Subsystems?

- a. JES1
- b. JES2
- c. JES3

The answers are JES2 and JES3.

2. What is the JCL statement to define a job step and job step-related information to the system?

The answer is EXEC.

3. True or False: A job can have a maximum of 256 job steps.

The answer is false. Only 255.

4. True or False: If a JCL syntax error is detected, the system bypasses the entire job.

The answer is true.

5. True or False: If a JCL execution time error is detected, the system bypasses the entire job.

The answer is false.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 2)

6. List two ways to submit JCL work in the system.

The answers are:

- a. Submit a batch
- b. Start a proc

7. Name three basic JCL statements.

The answers are:

- a. JOB
- b. EXEC
- c. DD

8. Valid return codes are in which of the following ranges:

- a. 0 – 4095
- b. 0 – 4096
- c. 0 – 8192

The answer is 0 - 4095.

9. True or False: JES2 and JES3 support the same syntax language.

The answer is false.

© Copyright IBM Corporation 2011

Unit 2

Checkpoint solutions (1 of 3)

1. Which of the following are valid JOB statement keyword parameters?

- a. REGION
- b. CLASS
- c. NOTIFY
- d. COMMAND

The answers are REGION, CLASS, and NOTIFY.

2. Which keyword defines the job log output class?

The answer is MSGCLASS.

3. True or False: Statements are coded from column 1 to column 72.

The answer is false; 1 to 71.

4. True or False: A continuation of a statement must start after column 16.

The answer is false; between 4 and 16.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 3)

5. True or False: STEP names must be unique in a JOB.
The answer is false. You can have duplicate names, although this is not recommended.
6. True or False: Key words can be in both uppercase and lowercase.
The answer is false; uppercase only.
7. True or False: REGION=0M means you can allocate as much as you want above 2 GB.
The answer is false; up to 2 GB.
8. True or False: MEMLIMIT=0 means you cannot allocate storage above 2 GB.
The answer is true.

© Copyright IBM Corporation 2011

Checkpoint solutions (3 of 3)

9. True or False: TIME=1440 means the job cannot execute longer than 24 hours (24x60).

The answer is false. TIME=1440 has the same meaning as TIME=NOLIMIT.

10. What is the DD statement to indicate data in the input stream?

The answer is //DD1 DD *.

11. Which keyword tells the system to automatically insert your user ID here so that the information will be sent to you?

- a. NOTIFY=SY&UID
- b. NOTIFY=&SYSID
- c. NOTIFY=&SYSUID

The answer is NOTIFY=&SYSUID.

12. Job name length can be:

- a. One to seven characters
- b. One to eight characters
- c. One to 10 characters

The answer is one to eight characters.

© Copyright IBM Corporation 2011

Unit 3

Checkpoint solutions (1 of 3)

1. What is the maximum length for a data set name?

- a. 24
- b. 40
- c. 44
- d. 48

The answer is 44.

2. True or False: UNIT=SYSDA is the same as UNIT=SYSALLDA.

The answer is false. SYSALLDA is a generic representing all DASD units, while SYSDA is an esoteric defined by the installation to cover some ranges of DASD units for allocation in JCL, with UNIT=SYSDA.

3. What does UNIT=/3390 mean?

- a. Allocate on unit number 3390
- b. Allocate on any 3390-type DASD

The answer is allocate on unit number 3390.

4. Which of the following is a valid statement?

- a. VOL=REF=DS3.JCL.CNTL
- b. VOL=SER=123456
- c. Both are valid statements.

The answer is both are valid statements.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 3)

5. Code the SPACE parameter to allocate 20 cylinders in primary and 10 in secondary allocation.

The answer is SPACE=(CYL,(20,10)).

6. In SPACE= (4096 , (300 , 100 , 50)) , what is the meaning of 50?

- a. 50 blocks of 4096 for secondary allocation
- b. 50 blocks of directory allocation

The answer is 50 blocks of directory allocation.

7. What does DISP=OLD mean?

- a. Data set exists
- b. Job requires exclusive access to data set
- c. The system starts writing at the end of the data set
- d. Both data set exists and job requires exclusive access to data set

The answer is both data set exists and job requires exclusive access to data set.

8. True or False: If DISP=MOD is coded for existing data set, records are appended to the logical end of data.

The answer is true.

© Copyright IBM Corporation 2011

Checkpoint solutions (3 of 3)

9. Name two ways by which the system can locate private libraries for program execution.

The answers are STEPLIB / JOBLIB / and System LINKLIST.

10. What is the meaning of DISP= (NEW, CATLG, DELETE) ?

The answer is: In the example, the status field specifies to create the data set (NEW); in the normal termination of the step to catalog (CATLG); and in abnormal termination to delete the data set (DELETE).

11. Which of the following are valid temporary data set names?

- a. DSN=&TEMP
- b. DSN=&&TEMP
- c. DSN=

The answers are DSN=&TEMP and DSN=&&TEMP.

© Copyright IBM Corporation 2011

Unit 4

Checkpoint solutions (1 of 2)

1. What is the name of the utility to manage VSAM data sets?
The answer is IDCAMS.

2. **True** or False: IEFBR14 can be used to delete an existing data set.
The answer is true.

3. True or **False**: IEBGENER can manage both sequential and PDS data sets.
The answer is false; only sequential.

4. Which utility can you use to copy a PDS?
 - a. IEBCOPY
 - b. IEBGENER
 - c. IDCAMS

The answers are IEBCOPY and IDCAMS.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 2)

5. With IEBGENER, what is the DD name for input data set?

- a. SYSIN
- b. SYSUT1
- c. SYSUT2

The answer is SYSUT1.

6. Name two system utilities (IEH) and their use.

The answers are IEHLIST (list and print VTOC and PDS directory) and IEHINIT (initialize tape).

7. How can you code the COND parameter to force the system to execute a following step?

The answer is COND=EVEN or COND=ONLY.

8. If a step ABENDs, the system bypasses:

- a. All following steps in the job
- b. The next step in the job

The answer is all following steps in the job.

© Copyright IBM Corporation 2011

Unit 5

Checkpoint solutions (1 of 4)

1. What is the maximum block size supported in z/OS?

- a. 4096
- b. 32756
- c. 32760

The answer is 32760.

2. With DCB= (BLKSIZE=3200 , LRECL=80 , RECFM=FB) , how many records do you have in each block?

The answer is 40.

3. With DCB= (BLKSIZE=3200 , LRECL=80 , RECFM=F) , how many records do you have in each block?

The answer is only one because RECFM=F not FB.

4. Which of the following are valid record formats?

- a. RECFM=F
- b. RECFM=FB
- c. RECFM=FV

The answers are RECFM=F and RECFM=FB.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 4)

5. How do you let the system use the optimum block size?

The answer is z/OS will calculate the optimum block size for a DASD data set if four conditions are true: 1) RECFM=FB or RECFM=VB (RECFM requests a blocked data set), 2) LRECL is coded, 3) the data set is sequential or partitioned (to be discussed later), and 4) BLKSIZE is omitted or specified to be zero.

6. What is the SMS JCL parameter to assign standard data set allocation attributes?

The answer is DATACLAS.

7. Name different data set types.

The answers are VSAM, non-VSAM, and UNIX files.

8. Name four VSAM organizations.

The answers are KSDS, ESDS, RRDS, and LDS.

9. True or False: In order for a data set to be SMS managed, the data set must have *all* of the following: data class, storage class, and management class.

The answer is false. The data set must have storage and management class, not data class.

© Copyright IBM Corporation 2011

Checkpoint solutions (3 of 4)

10. What are the AVGREC values to determine how the SPACE parameter's unit of allocation is interpreted?

The answers are K, M, and U.

11. Match DC, SC, MC, and SG to the following:

The answers are:

- a. MC: Defines backup and retention requirements
- b. DC: Defines model allocation characteristics for data sets
- c. SG: Creates logical groupings of volumes to be managed as a unit
- d. MC: Defines performance and availability goals

12. A UNIX file name can be up to how many characters long?

The answer is 255 (and 1023 for full path name).

13. True or False: A PDSE must be SMS-managed.

The answer is false.

14. True or False: An HFS must be SMS-managed.

The answer is false.

© Copyright IBM Corporation 2011

Checkpoint solutions (4 of 4)

15. True or False: A PDSE cannot contain executable programs.

The answer is false.

16. Name two utilities to allocate a zFS.

The answers are IDCAMS and TSO 'ALLOCATE' command.

17. True or False: To allocate a zFS, you must specify in JCL, DSNTYPE=ZFS.

The answer is false.

18. True or False: z/OS UNIX directory names and file names can be in uppercase, lowercase, or mixed case.

The answer is true.

19. What is the data set organization of zFS?

- a. VSAM
- b. LDS
- c. KSDS

The answers are VSAM and LDS.

© Copyright IBM Corporation 2011

Unit 6

Checkpoint solutions (1 of 2)

1. Which of the following have valid JCL GDG references to the current generation?
 - a. //GDGIN DD DSN=TEST.GDG(0),DISP=OLD
 - b. //GDGIN DD DSN=TEST.GDG(+0),DISP=OLD
 - c. Both //GDGIN DD DSN=TEST.GDG(0),DISP=OLD and //GDGIN DD DSN=TEST.GDG(+0),DISP=OLD

The answer is both //GDGIN DD DSN=TEST.GDG(0),DISP=OLD and //GDGIN DD DSN=TEST.GDG(+0),DISP=OLD.

2. Relative GDG numbers are not updated until end of:
 - a. Job
 - b. Step

The answer is job.

3. True or False: Once the job ends either normally or abnormally, the most recent entry in the catalog is now the +0 generation.

The answer is true.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 2)

4. If G0001V00 is the current generation and a new generation is being created with the JCL coded DSN=NAME.GDG (+3) , what would the new absolute name be?

The answer is G0004V00.

5. Match the following:

The answers are:

- NAME.GDG (+0) NAME.GDG.G0003V00 CURRENT GENERATION
- NAME.GDG (-1) NAME.GDG.G0002V00 NEXT OLDEST GENERATION
- NAME.GDG (-2) NAME.GDG.G0001V00 OLDEST GENERATION

© Copyright IBM Corporation 2011

Unit 7

Checkpoint solutions (1 of 3)

1. Which statement below is a valid statement to invoke a PROC?

- a. //STEP EXEC PROC=*procedure name*
- b. //STEP EXEC *procedure name*
- c. Both //STEP EXEC PROC=*procedure name* and //STEP EXEC *procedure name*.

The answer is both //STEP EXEC PROC=*procedure name* and //STEP EXEC *procedure name*.

2. What are the two types of procedures?

The answers are instream and cataloged.

3. Give a data set name where a cataloged procedure can be stored.

The answer is SYS1.PROCLIB.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 3)

4. True or False: Procedures can contain in stream SYSIN (//SYSIN DD *).

The answer is false.

5. True or False: DD statement parameters can be overridden, added, or nullified in any procedure step.

The answer is true.

6. What are the two statements required in an instream procedure (beginning and end)?

The answers are PROC and PEND.

7. True or False: DD statements which can be added with the modifying DD statements must appear last after all existing DD in a step.

The answer is false; they can be in any order.

© Copyright IBM Corporation 2011

Checkpoint solutions (3 of 3)

8. True or False: A symbolic parameter consists of an && followed by a name which is one to eight alphanumeric or national characters.
The answer is false; only a single ampersand (&).
9. True or False: Default values must be assigned to each of the symbolic parameters on the PROC statement at the beginning of the procedure.
The answer is false. Defaults can be left empty.
10. Assuming a proc defined as //PROCSMP PROC P1=, P2=, what is the valid way to call this proc?
 - a. //STEP1 EXEC PROCSMP, P1=PARM1, P2=PARM2
 - b. //STEP1 EXEC PROCSMP, P2=PARM1, P1=PARM2
 - c. Both //STEP1 EXEC PROCSMP, P1=PARM1, P2=PARM2 and //STEP1 EXEC PROCSMP, P2=PARM1, P1=PARM2.

The answer is both //STEP1 EXEC PROCSMP, P1=PARM1, P2=PARM2 and //STEP1 EXEC PROCSMP, P2=PARM1, P1=PARM2.

© Copyright IBM Corporation 2011

Unit 8

Checkpoint solutions (1 of 2)

1. Which of the following are valid statements for input data set with IEBCOPY?

- a. [//IN DD DSN=...](#)
- b. [//SYSIN DD DSN=...](#)
- c. [//SYSUT1 DD DSN=...](#)
- d. All of them

The answers are [//IN DD DSN=...](#) and [//SYSUT1 DD DSN=...](#).

2. With the IEBCOPY input statement, what is the meaning of the R in the following example: SELECT MEMBER= ((NAME1, NAME2, R))

The answer is [copy NAME1 as NAME2, and replace NAME2 if it already exists.](#)

3. IEBCOPY can be used to:

- a. [Compress a PDS](#)
- b. [Rename members of a PDS](#)
- c. [Replace members of a PDS](#)
- d. Delete members of a PDS
- e. All of the above

The answers are [compress a PDS, rename members of a PDS, and replace members of a PDS.](#)

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 2)

4. True or False: IEHPROGM is the recommended utility to delete/scratch data sets.

The answer is false. IDCAMS is the recommended utility.

5. What utility can be used to update and modify sequential data sets and members of partitioned data sets?

The answer is IEBUPDTE.

6. What is the name of the input DD statement when you execute UNIX shell scripts in batch?

The answer is STDIN.

© Copyright IBM Corporation 2011

Unit 9

Checkpoint solutions (1 of 2)

1. What is the JCL statement required to support include groups?

- a. JCLIN
- b. INJCL
- c. JCLINPUT
- d. JCLLIB

The answer is JCLLIB.

2. True or False: An INCLUDE group must contain at least one EXEC statement.

The answer is false.

3. How many member names can be coded in an INCLUDE statement?

- a. One
- b. Eight
- c. No limit

The answer is one.

4. True or False: An INCLUDE group can contain a PROC.

The answer is false.

5. True or False: A PROC can contain an INCLUDE statement.

The answer is true.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 2)

6. True or False: A SET statement must appear right after the JOB statement.
The answer is false.
7. True or False: You can use static symbols like &SYSNAME in batch JCL.
The answer is false; only in started tasks JCL (STC).
8. Multiple SET statements can assign a value to a symbolic parameter. In this case, the value assigned on which SET statement is used?
 - a. First
 - b. Last**The answer is last.**
9. Which of the following is a valid system static symbol?
 - a. &SYSUID
 - b. &SYSCLONE
 - c. &JOBNAME**The answer is &SYSCLONE.**

© Copyright IBM Corporation 2011

Unit 10

Checkpoint solutions (1 of 2)

1. True or False: The COND parameter on the first step of a job is ignored.
The answer is true.
2. True or False: The IF construct on the first step of a job is ignored.
The answer is false.
3. What are the constructs required after an IF?
 - a. THEN
 - b. ELSE
 - c. ENDIFThe answers are THEN and ENDIF.
4. What is the meaning of // IF (step1. ¬ABEND)?
The answer is if step1 did not abend.
5. True or False: You must specify the step name for an IF condition.
The answer is false.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 2)

6. True or False: If you have not given the step name, the return code from the last step is used for checking.
The answer is false; for all previous steps.
7. True or False: Either the THEN clause or ELSE clause must contain at least one EXEC statement.
The answer is true.
8. True or False: The action following THEN can be omitted.
The answer is true.
9. Give another syntax for IF (ABEND | STEP1.RC > 8).
The answer is IF (ABEND OR STEP1.RC GT 8) THEN.
10. True or False: If a parameter is coded on both the DD and the OUTPUT statements, the parameter on the DD statement will be ignored.
The answer is false. The parameter on the SYSOUT DD statement will be used.

© Copyright IBM Corporation 2011

Unit 11

Checkpoint solutions (1 of 2)

1. Which of the following can DFSORT perform:

- a. Sort records
- b. Merge records
- c. Copy records
- d. All of the above

The answer is all of the above.

2. True or False: Fields are sorted from right to left as they appear in the FIELDS operand.

The answer is false; left to right.

3. What is the maximum number of SORTWK data sets?

The answer is 255.

4. Name three types of SORTWK intermediate work space that DFSORT can use for sorting.

The answers are SORTWK data sets, hiperspace, and memory objects (virtual storage above bar 2 GB line).

5. What is the maximum number input of data sets that can be merged by SORT?

The answer is 100.

© Copyright IBM Corporation 2011

Checkpoint solutions (2 of 2)

6. What is the name of the input DD statement used for merge?

The answer is SORTINnn (00 to 99).

7. Explain the following sort job:

```
//SORTIT EXEC PGM=SORT,PARM='ABEND'  
//SORTIN DD * number in column 1, name in 10, state in 19  
9283873   JOAN      CT  
70232322  JANE      CN  
8432343   MARIO     MA  
5549023   JILL      CT  
6998781   JENNIFER VT  
/*  
//SORTOUT DD SYSOUT=*  
//SYSIN   DD *  
        SORT FIELDS=COPY,SKIPREC=3,STOPAFT=2
```

The answer is copy, starting with fourth record, and stop after printing two records.

© Copyright IBM Corporation 2011

Unit 12

Checkpoint solutions

1. What is the maximum volume count for a data set?

- a. 8
- b. 59
- c. 60

The answer is 59.

2. How many volumes will be allocated with
UNIT=(3390,2), VOL=(, , ,4) ?

The answer is two.

3. If you want to use a 128-track tape, how would you code the UNIT parameter?

The answer is UNIT=3590-1.

4. What is the meaning of LABEL=(3,SL) ?

The answer is request the third data set on a standard labeled tape.

© Copyright IBM Corporation 2011

Appendix B. JCL examples

JCL: Useful sample JCLs for any use

- JCL examples of DD statements for existing data
- Sample DD statements for creating data
- Examples of GDGs
- Defining VSAM
- Examples of IDCAMS to print
- Examples of IDCAMS to copy
- Examples of running TSO in batch, running REXX
- Examples of ISPF compare in batch.
- Example of a safe compress
- Example of copying a PDS
- IEFBR14
- AMBLIST: Show load module attribute
- SORT examples

JCL examples of DD statements for existing data

Open for INPUT to READ. Open for OUTPUT to completely replace VSAM
//ddname DD DSN=dataset-name,DISP=SHR

Example:

```
//INFILE DD DSN=MY.DATASET.DATA,DISP=SHR
```

"SEQUENTIAL"

File is not present.

```
//ddname DD DUMMY,  
//      DCB=(LRECL=rec-len,BLKSIZE=blk-size,RECFM=rec-format)
```

Example:

```
//INFILE DD DUMMY,  
//      DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
```

In-stream data. 80 characters, fixed format

```
//ddname DD *  
data goes here  
/*
```

Example:

```
//INFILE DD *  
Linda  
Ellen  
/*
```

Disk or Tape non-gdg, non-pds member.

```
//ddname DD DSN=dataset-name,DISP=SHR
```

Example:

```
//INFILE DD DSN=MY.DATASET.DATA,DISP=SHR
```

Disk or tape, gdg.

```
//ddname DD DSN=dataset-name(0),DISP=SHR
```

Example:

```
//INFILE DD DSN=MY.DATASET.DATA(0),DISP=SHR
```

Disk, existing PDS member.

```
//ddname DD DSN=dataset-name(member-name),DISP=SHR
```

Example:

```
//INFILE DD DSN=MY.DATASET.DATA(M1),DISP=SHR
```

Temporary disk. Previously Passed

```
//ddname DD DSN=&&dataset-name,DISP=(OLD,DELETE) or OLD,PASS
```

Example:

```
//INFILE DD DSN=&&TEMP,DISP=(OLD,DELETE)
```

Sample DD Statements for Creating data

Open for OUTPUT only

VSAM

```
//ddname DD DSN=dataset-name,DISP=SHR
```

Example:

```
//INFILE DD DSN=MY.DATASET.DATA,DISP=SHR
```

"SEQUENTIAL"

Throw away the file.

```
//ddname DD DUMMY,  
//      DCB=(LRECL=rec-len,BLKSIZE=blk-size,RECFM=rec-format)
```

Example:

```
//INFILE DD DUMMY,  
//      DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
```

Spooled to printer

```
//ddname dd SYSOUT=printer-class
```

Example:

```
//OUTFILE DD SYSOUT=*
```

Disk, non-gdg, non-pds member.

```
//ddname DD DSN=dataset-name,DISP=(NEW,CATLG,DELETE),
```

```
//          UNIT=diskunit,SPACE=(TRK,1)
```

Example:

```
//OUTFILE DD DSN=MY.DATASET.DATA,DISP=(NEW,CATLG,DELETE),  
//          UNIT=SYSDA,SPACE=(TRK,1)
```

Tape, non-gdg.

```
//ddname DD DSN=dataset-name,DISP=(NEW,CATLG,DELETE),  
//          UNIT=tapeunit  
/* other parameters may be required by your company
```

Example:

```
//ddname DD DSN=MY.DATASET.DATA,DISP=(NEW,CATLG,DELETE),  
//          UNIT=TAPE  
/* other parameters may be required by your company
```

Disk, sequential gdg.

```
//ddname DD DSN=dataset-name(+1),DISP=(NEW,CATLG,DELETE),  
//          UNIT=diskunit,SPACE=(TRK,1),  
//          DCB=model-dscb optional at some companies
```

Example:

```
//OUTFILE DD DSN=MY.DATASET.DATA(+1),DISP=(NEW,CATLG,DELETE),  
//          UNIT=SYSDA,SPACE=(TRK,1),  
//          DCB=MODEL.DSCB optional at some companies. find its name
```

Disk, new or existing PDS member. Existing member will be overwritten

```
//ddname DD DSN=dataset-name(member-name),DISP=SHR
```

Example:

```
//OUTFILE DD DSN=MY.DATASET.DATA(M1),DISP=SHR
```

Temporary disk..

```
//ddname DD DSN=&&dataset-name,DISP=(NEW,PASS,DELETE),  
//          UNIT=diskunit,SPACE=(TRK,1)
```

Example:

```
//OUTFILE DD DSN=&&TEMP,DISP=(NEW,PASS,DELETE),  
//          UNIT=SYSDA,SPACE=(TRK,1)
```

Temporary disk. Not used again in same job. A work file.

```
//ddname DD UNIT=diskunit,SPACE=(TRK,1)
```

Example:

```
//OUTFILE DD UNIT=SYSDA,SPACE=(TRK,1)
```

Tape, gdg.

```
//ddname DD DSN=dataset-name(+1),DISP=(NEW,CATLG,DELETE),  
//           UNIT=tapeunit,  
//           DCB=model-dscb optional at some companies
```

Example:

```
//OUTFILE DD DSN=MY.DATASET.DATA(+1),DISP=(NEW,CATLG,DELETE),  
//           UNIT=TAPE,  
//           DCB=MODEL.DSCB optional at some companies. find out it
```

Examples of GDGs

```

/* GDG1DEF DEFINE A GENERATION DATA GROUP
//GDG1DEF EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DEFINE GDG (NAME(userid.SAMPLE.GDG.BASE) -
  LIMIT(10) NOEMPTY SCRATCH)
/*
/*GDG2MAKE ACTUALLY CREATE A NEW G. D. SET
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD *
LINDA
ELENA
/*
//SYSUT2 DD DSN=userid.SAMPLE.GDG.BASE(+1),DISP=(NEW,CATLG),
//           SPACE=(TRK,1),
//           DCB=model-dscb optional at some companies

/* GDG3ALT CHANGE A GENERATION DATA GROUP'S DEFINITION
//GDG3ALT EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  ALTER userid.SAMPLE.GDG.BASE -
  LIMIT(15)
/*
/* GDG4DEL THIS WILL DELETE ALL THE MEMBERS OF THE GROUP
//ADIOS EXEC PGM=IEFBR14
//BYEBYE DD DSN=userid.SAMPLE.GDG.BASE,DISP=(OLD,DELETE)

/* GDG5DEL THIS WILL REMOVE THE GD GROUP FROM THE CATALOG
/* BE SURE TO DO GDG4DEL FIRST
//GDG5DEL EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DELETE userid.SAMPLE.GDG.BASE GDG
/*

```

Defining VSAM

```
/*      DEFINE A VSAM KSDS
//DEFKSDS  EXEC PGM=IDCAMS
//SYSPRINT DD    SYSOUT=*
//SYSIN    DD    *
/* the trailing - is for continuation */
/* if delete fails there will be an error message which you can ignore */
DELETE (userid.VSAM.KSDS) CLUSTER

DEFINE CLUSTER -
  NAME(userid.VSAM.KSDS) -
  CYLINDERS(1,1) -
  KEYS(10,0) /* length displacement */ -
  RECORDSIZE(80,80) /* avg max */ -
  INDEXED) /* ksds */

/*
Some options for defining VSAM
RECOVERY          Allow a restart of a load
SPEED             Opposite of recovery
UNIQUE            The cluster will not be part of a data space containing other
                  clusters
SUBALLOCATE       Opposite of unique
REPLICATE         Copy each index record on the track as many times as it will fit
NOREPLICATE       Opposite of unique
REUSE              If there is data in the cluster, a repro or load will clobber the
                  existing data
NOREUSE            Can do only one repro or load on this cluster

SHAREOPTIONS      Region   system
```

Region means roughly "this CICS" (an MVS can have more than one CICS running)

- | | |
|---|--|
| 1 | Many jobs can read at same time |
| 2 | Many jobs can read, only one can write |
| 3 | Many jobs can read, many jobs can update, you're on
your own regarding data integrity |
| 4 | Many jobs can read, many jobs can update |

System means roughly "this MVS"

- | | |
|---|--|
| 3 | Many jobs can read, many jobs can update, you're on
your own regarding data integrity |
|---|--|

```
/* DEFINE A VSAM ESDS
//DEFESDS EXEC PGM=IDCAMS
//SYSPRINT DD SYOUT=*
//SYSIN DD *
DELETE (userid.VSAM.ESDS) CLUSTER

DEFINE CLUSTER +
NAME(userid.VSAM.ESDS) -
CYLINDERS(1,1) -
RECORDSIZE(80,80) -
NONINDEXED)

/* DEFINE A VSAM RRDS
//DEFRRDS EXEC PGM=IDCAMS
//SYSPRINT DD SYOUT=*
//SYSIN DD *
DELETE (userid.VSAM.RRDS) CLUSTER

DEFINE CLUSTER -
NAME(userid.VSAM.RRDS) -
CYLINDERS(1,1) -
RECORDSIZE(80,80) -
NUMBERED)
/*
/* LOAD A VSAM DATASET (KSDS, ESDS, RRDS) THEN PRINT IT
//LOADVSAM EXEC PGM=IDCAMS
//SYSPRINT DD SYOUT=*
//SYSIN DD *
REPRO INFILE(INDD) OUTDATASET(userid.VSAM.xxxx)

PRINT INDATASET(userid.VSAM.xxxx) CHARACTER

/*
//INDD DD *      for KSDS, records must be in order
ELENA
LINDA
NADIA
SUSAN
/*
```

Examples of IDCAMS to print

General JCL to print a file (sequential, VSAM, PDS member)

```
//STEP1 EXEC PGM=IDCAMS  
//SYSPRINT DD SYSOUT=*  
//INFILE      DD DSN=name of file to be printed,DISP=SHR  
//SYSIN      DD *  
          PRINT INFILE(INFILE) CHARACTER  
/*
```

Example of JCL to print the file ABCCO.TEST.DATA (could be sequential, VSAM, PDS member)

```
//STEP1 EXEC PGM=IDCAMS  
//SYSPRINT DD SYSOUT=*  
//INFILE      DD DSN=ABCCO.TEST.DATA,DISP=SHR  
//SYSIN      DD *  
          PRINT INFILE(INFILE) CHARACTER  
/*
```

Notes: change CHARACTER to DUMP in order to print in both character and hex
Change CHARACTER to HEX in order to print in hex

Examples of IDCAMS to copy

General JCL to copy a file (sequential, VSAM, PDS member) to an existing file (sequential, VSAM, PDS member)

```
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//INFILE      DD DSN=name of file to be copied,DISP=SHR
//SYSIN DD *
      REPRO INFILE(INFILE) OUTFILE(OUTFILE)
/*
//OUTFILE      DD DSN=name of file to be replaced,DISP=SHR
```

Example of JCL to copy the file ABCCO.TEST.DATA (could be sequential, VSAM, PDS member) to the existing file ABCCO.SAMPLE.DATA

```
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//INFILE      DD DSN=ABCCO.TEST.DATA,DISP=SHR
//SYSIN DD *
      REPRO INFILE(INFILE) OUTFILE(OUTFILE)
/*
//OUTFILE      DD DSN=ABCCO.SAMPLE.DATA,DISP=SHR
```

Notes:

IDCAMS will copy any of the following types:

- Sequential
- PDS member
- VSAM
- ISAM

to any of the following:

- Sequential
- PDS member
- VSAM
- ISAM

This JCL will clobber any data that already exists in the output file.

However, if the output file is VSAM and you want to clobber it, you must add the keyword REUSE, for example:

REPRO INFILE(INFILE) OUTFILE(OUTFILE) REUSE

and you must have defined the VSAM cluster with REUSE.

Using IDCAMS to delete a file. If the file is not found,

there will be an error message and high return code.

```
/* delete with idcams
//DELETE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
DELETE           dataset-name
```

Using IDCAMS to shorten a file by copying it with the skip or count parameter.

```
/* shorten a file
//SHORT01 EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
      REPRO INFILE(INDD) OUTFILE(OUTDD) SKIP(0001000) COUNT(0001000)
/*
//INDD    DD  DSN=input-file,DISP=OLD
//OUTDD   DD  DSN=output-file,
//          DISP=(NEW,CATLG,DELETE) ,
//          SPACE=(TRK,(5,2),RLSE) ,
//          DCB=(RECFM=FB,LRECL=xx)
/***
```

Examples of running TSO in batch, running REXX

You can execute TSO commands, CLISTS, REXX programs.

```
/* Run TSO in batch
//TSOBATCH EXEC PGM=IKJEFT1A,DYNAMNBR=200
//SYSPROC DD DSN=your-id.clist,DISP=SHR optional
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
PROFILE PREFIX(your-id)
LISTCAT LEVEL(your-id)
/*
```

Using TSO in batch to delete a file.

```
/* Run TSO in batch
/* also IKEFFT01, IKJEFT1B
//TSOBATCH EXEC PGM=IKJEFT1A,DYNAMNBR=200
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
PROFILE PREFIX(your-id)
DELETE 'dataset-name'
```

Running a REXX program in batch

Obtain the name of your REXX library and fill it in.
If your REXX program reads a file, you can put in a DD for
the file and avoid having to do an ALLOC in the program.

```
//RUNREXX EXEC PGM=IKJEFT1A,DYNAMNBR=200
//SYSEXEC DD DSN=dsn-of-rexx-library,DISP=SHR
//INFILE   DD DISP=SHR,DSN=dsn-of-input-file
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
///*
//SYSTSIN  DD *
PROFILE PREFIX(O054)
%MYREXX
```

Examples of ISPF compare in batch

```
/*SPF compare in batch
//SUPER1 EXEC PGM=ISRSUPC,
//           PARM=(DELTAL,LINECMP,'','')
//OLDDD DD DSN=old-dataset,
//           DISP=SHR
//NEWDD DD DSN=new-dataset,
//           DISP=SHR
//OUTDD DD SYSOUT=*
/*
```

The ISPF Search. Lower overhead, won't tie up your terminal during search.

```
/* spf search option 3.14
//SEARCH EXEC PGM=ISRSUPC,PARM=(SRCHCMP,'ANYC')
//NEWDD DD DSN=library-to-be-searched,
//           DISP=SHR
//OUTDD DD SYSOUT=*
//SYSIN DD *
SRCHFOR 'character-string-to-be-searched-for'
SELECT MEMBER1, MEMBER2 optional select. restricts search to these members
SELECT MEMBER3, MEMBER4
/*
```

Example of a safe compress

The world's safest compress for PDS's. Can't destroy your library. Creates a backup first. If the compress works, it deletes the backup. If the compress fails, it keeps the backup.

```
/** the ultimate batch compress
/** creates a backup library first.
/** then deletes the backup last, if all worked
//ERASE EXEC PGM=IEFBR14
//OUTPDS DD DSN=your-id.name-of-backup-lib,DISP=(MOD,DELETE),
//          UNIT=SYSDA,SPACE=(TRK,0)
/**
//COPY EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COPY INDD=INPDS,OUTDD=OUTPDS
/*
//INPDS DD DSN=library-to-be-compressed,DISP=SHR
//OUTPDS DD DSN=your-id.name-of-backup-lib,REFDD=*.INPDS,
//          UNIT=SYSDA, or your disk unit,
//          DISP=(NEW,CATLG,DELETE),
//          SPACE=(TRK,(100,10,50),RLSE)
/**
/** produce jcl error if the previous step failed
//JCLOUT EXEC PGM=IEFBR14,COND=(0,EQ,COPY)
//DD1 DD DSN=NOT.THERE,DISP=SHR
/**
/** this will do the compress
//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COPY INDD=INPDS,OUTDD=INPDS
/*
//INPDS DD DSN=library-to-be-compressed,DISP=SHR
/**
/** if compress fails, send tso message
//          IF (RC GT 0) THEN
//TSOBATCH EXEC PGM=IKJEFT1A,DYNAMNBR=200,COND=(0,LT)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
PROFILE PREFIX(your-id)
SEND 'COMPRESS FAILED library-to-be-compressed' USER(your-id) LOGON WAIT
```

```
//      ELSE
//ERASE   EXEC PGM=IEFBR14,COND=(0,LE)
//OUTPDS  DD DSN=your-id.name-of-backup-lib,DISP=(MOD,DELETE) ,
//          UNIT=SYSDA,SPACE=(TRK,0)
//      ENDIF
```

Example of copying a PDS

```

//* COPY ONE PDS TO ANOTHER DO NOT REPLACE LIKE-NAMED MEMBERS
//LIBCOPY EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(3,3))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(3,3))
//SYSUT4 DD UNIT=SYSDA,SPACE=(CYL,(3,3))
//INLIB DD DSN=library-containing-members,DISP=SHR
//OUTLIB DD DSN=library-to-put-members-in,DISP=SHR
//SYSIN DD *
      COPY INDD=INLIB,OUTDD=OUTLIB
/*
//* OPTIONAL THINGS:
//*
//* TO COPY BUT EXCLUDE TWO MEMBERS:
//* COPY INDD=INLIB,OUTDD=OUTLIB
//* EXCLUDE MEMBER=(m1,m2)
//*
//* TO COPY AND REPLACE:
//* COPY INDD=((INLIB,R)),OUTDD=OUTLIB

//*iebcopy to copy a pds
//*in: pds
//*out: pds
//copy only members named MEM1 and MEM2
//do not replace those members if they exist in the output pds
//IEBCOPY EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
      COPY INDD=(INPDS),OUTDD=OUTPDS
      SELECT MEMBER=MEM1
      SELECT MEMBER=MEM2
/*
//*
//*
//*
//*iebcopy to copy a pds
//*in: pds
//*out: pds
//copy only members named MEM1 and MEM2
//replace those members if they exist in the output pds
//IEBCOPY EXEC PGM=IEBCOPY

```

```
//SYSPRINT DD SYSOUT=*
//SYSIN      DD   *
COPY INDD=((INPDS,R)),OUTDD=OUTPDS
SELECT MEMBER=MEM1
SELECT MEMBER=MEM2
/*
```

IEFBR14

Using IEFBR14 to delete a file. If the file is not found, there will be no error message and no high return code.

```
//ERASE      EXEC PGM=IEFBR14
//OUTPDS    DD DSN=your-id.name-of-backup-lib,DISP=(MOD,DELETE),
//           UNIT=SYSDA,SPACE=(TRK,0)
//*
```

Using IEFBR14 to avoid deleting and reallocating a file. If you run a job frequently, and it needs the same output dataset each time, you can use this JCL.

It finds the dataset, if it already exists. Then it clobbers it, avoiding the overhead of reallocating. Note that the disposition on the step that actually uses the file, later in the job, must be (OLD,CATLG)

If the dataset doesn't exist, it is created.

```
//FINDPASS EXEC PGM=IEFBR14
//DD1       DD   DSN=dsn-to-be-used-later-in-job,
//           UNIT=SYSDA,SPACE=(TRK,(5,5)),DISP=(MOD,PASS),
//           LRECL=133,RECFM=FB,BLKSIZE=27930
//*
```

AMBLIST: Show load module attributes

```
/* SHOW LOAD MODULE ATTRIBUTES
//AMBLIST EXEC PGM=AMBLIST
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
LISTIDR DDN=SYSLMOD, MEMBER=program-name GIVES ATTRIBUTES

LISTLOAD DDN=SYSLMOD, MEMBER=program-name GIVES ATTRIBUTES AND CONTENTS
/*
//SYSLMOD DD DSN=library-the-program-is-in,DISP=SHR
```

SORT examples

```

///* to sort
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
* sort by name
  SORT FIELDS=(10,9,CH,A),DYNALLOC=(SYSDA,6),FILSZ=E2000
/*
///* END OF JOB

///* sort and do totals
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
* sort by state
* sum phone number
  SORT FIELDS=(19,2,CH,A),DYNALLOC=(SYSDA,6),FILSZ=E2000
  SUM FIELDS=(1,7,ZD)
* ZD is zoned decimal. signed or unsigned
* PD packed decimal. BI Binary
/*
///* END OF JOB

```

```
/* to sort and shift fields
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD * number in column 1, name in 10, state in 19
* sort by name (name is now in 1)
  SORT FIELDS=(01,09,CH,A),DYNALLOC=(SYSDA,6),FILSZ=E2000
  INREC FIELDS=(01:10,09,      move name from 10 to 1
                  10:01,07)    move number from 1 to 10
* inrec moves data before sorting starts
/*
///* END OF JOB
```

```
/* to sort and shift fields
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD *   number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
* sort by name
  SORT FIELDS=(10,09,CH,A),DYNALLOC=(SYSDA,6),FILSZ=E2000
  OUTREC FIELDS=(01:10,09,      move name from 10 to 1
                  10:01,07)    move number from 1 to 10
* outrec moves data after sorting ends
/*
///* END OF JOB
```

```
/* PLAIN COPY
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
```

```

//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
  SORT FIELDS=COPY
/*
///* END OF JOB

///*      COPY BUT SKIP 3 RECORDS
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
  SORT FIELDS=COPY,SKIPREC=3
/*
///*      COPY. START WITH 4TH RECORD. STOP AFTER PRINTING 2 RECORDS
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
  SORT FIELDS=COPY,SKIPREC=3,STOPAFT=2  STOP AFTER PRINTING 2

```

```
/*
///* COPY BUT ONLY IF LITERAL CT IS IN COLUMN 19 OF DATA
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER  VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
      SORT FIELDS=COPY           COPY BUT
      INCLUDE COND=(19,2,CH,EQ,C'CT') ONLY IF CT IN COL 19
/*
///*
///* COPY BUT NOT IF LITERAL CT IS IN COLUMN 19 OF INPUT
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER  VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
      SORT FIELDS=COPY           COPY BUT
      OMIT      COND=(19,2,CH,EQ,C'CT') NOT IF CT IN COL 19
/*
///*
///* COPY. INCLUDE RECORDS WITH LITERAL CT IN COLUMN 19,
///*      OR LITERAL J IN COLUMN 10
//SORTIT EXEC PGM=SORT,PARM='ABEND'
```

```

//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
  SORT FIELDS=COPY           USING AN OR
*                   COPY IF CT IN COL 19
*                   OR J IN COL 10
 INCLUDE COND=(19,2,CH,EQ,C'CT',OR,10,1,CH,EQ,C'J')
* AND WORKS TOO. ALSO X'C1' FOR A
/*

```

```

/** COPY INPUT FILE TO OUTPUT FILE BUT MOVE AS FOLLOWS
/** PUT THIS IN OUTPUT COLUMN 20: 9 CHARACTERS STARTING IN COLUMN 1
/**                                     COLUMN 40: 16 CHARACTERS STARTING IN COLUMN 10
//SORTIT EXEC PGM=SORT,PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
* COPY INPUT FILE TO OUTPUT FILE BUT MOVE AS FOLLOWS
* PUT THIS IN OUTPUT COLUMN 20: 9 CHARACTERS STARTING IN COLUMN 1
*                                     COLUMN 40: 16 CHARACTERS STARTING IN COLUMN 10
  SORT FIELDS=COPY
  INREC FIELDS=(20:1,09,40:10,16)
/*
/** COPY INPUT FILE TO OUTPUT FILE BUT MOVE AS FOLLOWS
/** PUT THIS IN OUTPUT COLUMN 20: 9 CHARACTERS STARTING IN COLUMN 1

```

```
/* COLUMN 40: 16 CHARACTERS STARTING IN COLUMN 10
/* include only records from CT
//SORTIT EXEC PGM=SORT, PARM='ABEND'
//SYSOUT DD SYSOUT=*
//SORTIN DD * number in column 1, name in 10, state in 19
9283873 JOAN      CT
7023232 JANE      CN
8432343 MARIO     MA
5549023 JILL      CT
6998781 JENNIFER  VT
8432343 MURIEL    RI
/*
//SORTOUT DD SYSOUT=*
//SYSIN   DD *
* COPY INPUT FILE TO OUTPUT FILE BUT MOVE AS FOLLOWS
* PUT THIS IN OUTPUT COLUMN 20: 9 CHARACTERS STARTING IN COLUMN 1
*                                COLUMN 40: 16 CHARACTERS STARTING IN COLUMN 10
SORT FIELDS=COPY
INREC FIELDS=(20:1,09,40:10,16)
INCLUDE COND=(19,2,CH,EQ,C'CT') only if CT in column 19
* include condition looks at input columns before INREC or OUTREC work
/*
```

```
/*
/*syncsort to eliminate duplicates
/* in: seq
/* out: seq
//SYNCSORT EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DISP=SHR,      <-the input
//          DSN=the.input.dsn
//SORTOUT DD DISP=(NEW,CATLG), <-the output
//          UNIT=SYSDA,LIKE=the.input.dsn,
//          SPACE=(TRK,(1,1),RLSE)
//SYSIN   DD *
      SORT FIELDS=(1,80,CH,A)
      SUM FIELDS=NONE
/*
/*
/*
/*
/*syncsort to include records on a condition
```

```
/*uses OUTREC to construct the output record
/*from fields on the input record
/* in: seq
/* out: seq
//SYNCSORT EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DISP=SHR,      <- the input
//          DSN=the.input.dsn
//SORTOUT DD DISP=(NEW,CATLG), <- the output
//          UNIT=SYSDA,LIKE=the.input.dsn,
//          SPACE=(TRK,(1,1),RLSE)
//SYSIN DD *
      SORT FIELDS=(1,80,CH,A)
      INCLUDE COND=((51,7,CH,EQ,C'STRING1'),OR,
                     (51,7,CH,EQ,C'STRING2'))
      OUTREC FIELDS=(01:11,08,
                      20:51,19)
* a literal may be used in the outrec:
* OUTREC FIELDS=(C'!',
*                 20:C'ABCD')
      SUM FIELDS=NONE
/*
/***
/***
```


Bibliography

Manuals:

SA22-7597	<i>z/OS MVS JCL Reference</i>
SC26-7414	<i>DFSMS/dfp Utilities</i>
SA22-7626	<i>z/OS MVS System Codes</i>
SA22-7598	<i>z/OS MVS JCL User's Guide</i>
SC26-7394	<i>DFSMS/dfp Access Method Services for the Integrated Catalog Facility</i>
SC26-7412	<i>DFSMS/dfp Using Magnetic Tapes</i>
SC26-7523	<i>DFSORT Application Programming Guide</i>
SC26-7527	<i>Getting Started with DFSORT</i>
SA22-7631	<i>z/OS MVS System Messages, Vol. 1</i>
SA22-7632	<i>z/OS MVS System Messages, Vol. 2</i>
SA22-7633	<i>z/OS MVS System Messages, Vol. 3</i>
SA22-7634	<i>z/OS MVS System Messages, Vol. 4</i>
SA22-7635	<i>z/OS MVS System Messages, Vol. 5</i>
SA22-7636	<i>z/OS MVS System Messages, Vol. 6</i>
SA22-7637	<i>z/OS MVS System Messages, Vol. 7</i>
SA22-7638	<i>z/OS MVS System Messages, Vol. 8</i>
SA22-7639	<i>z/OS MVS System Messages, Vol. 9</i>
SA22-7640	<i>z/OS MVS System Messages, Vol. 10</i>

Web URLs:

<http://www-03.ibm.com/servers/eserver/zseries/zos/bkserv/>

Online version of z/OS reference manuals
in PDF and BookManager format

IBM
®