

PUBLICATIONS REVISION
System 80
OS/3 Consolidated Data Management Programming Guide
UP-9978 Rev. 1

This Library Memo announces the release and availability of the *System 80 OS/3 Consolidated Data Management Programming Guide*, UP-9978 Rev. 1.

This guide is a standard library item (SLI). It is part of the standard library provided automatically with the purchase of the product.

This guide explains what consolidated data management is, how it works, the data structures that are involved, and how errors and exceptions are handled. Additionally, it describes the formats and conventions for the various types of files and discusses those things you must consider when using each type of file in your program.

Changes to this guide for Release 12.0:

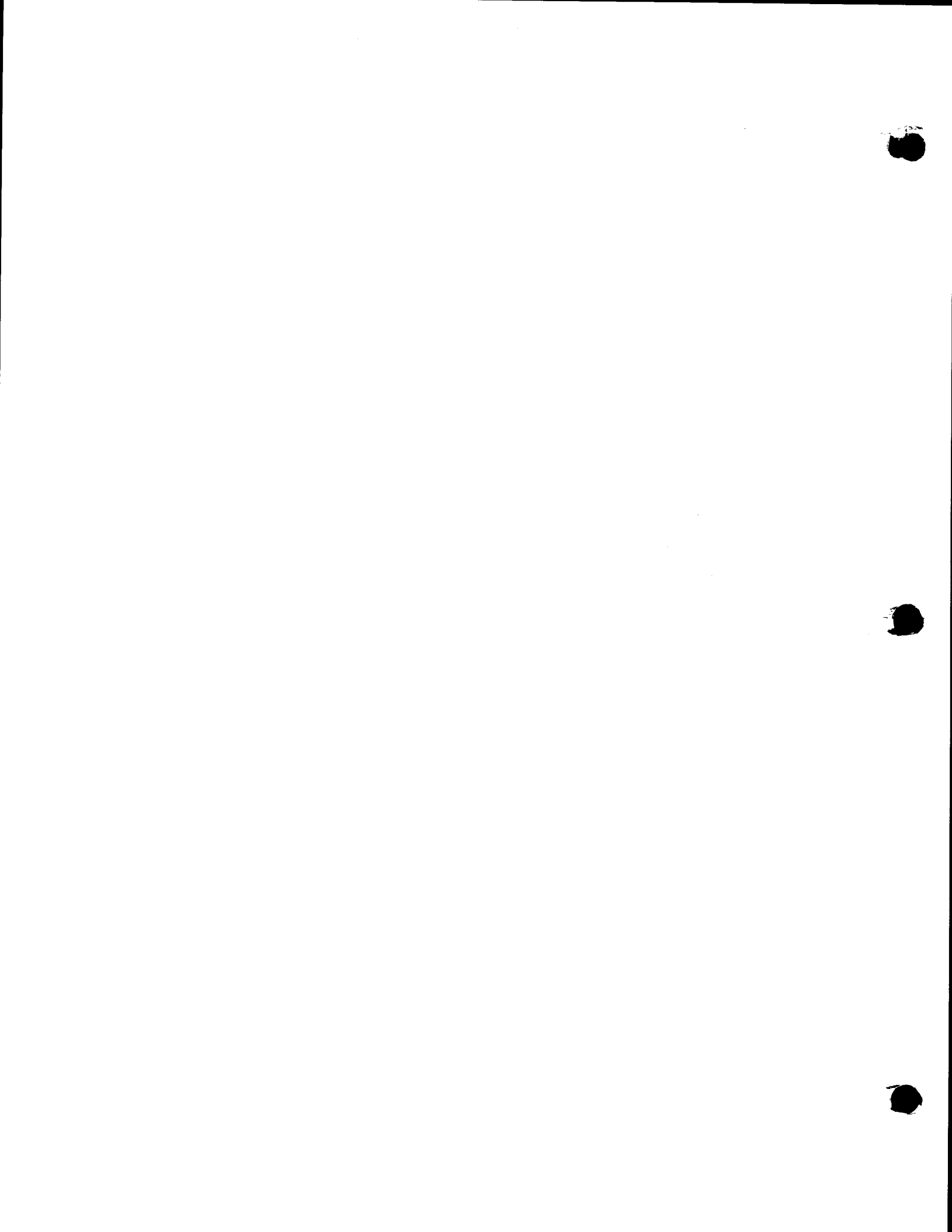
- The DMRECV=YES SUPGEN parameter does not provide recovery for temporary work files or system files as it did in prior releases.
- // DD RECV=OFF, a new job control language specification that allows you to temporarily "turn off" recovery.
- // DD MSGSUPP, a new job control language specification that allows you to suppress the DM36 DUPLICATE RECORD message, to suppress the LB05 MODULE NOT FOUND message, or to suppress all data management messages.
- // DD RESTORE=YES, a new job control language specification that allows you to restore your file if it is unintentionally initialized.

All other changes in this guide are corrections, deletions, or expanded descriptions applicable to items present in the software prior to this release.

Additional copies may be ordered through your local Unisys representative.

Destruction Notice: This revision supersedes and replaces the *OS/3 Consolidated Data Management Concepts and Facilities*, UP-9978, released on Library Memo dated September 1984. Please destroy all copies of UP-9978 and its Library Memo.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists MBZ, MCZ, and MMZ	Mailing Lists MB00, MB01, MBW, M28U, and M29U (142 pages plus Memo)	Library Memo for UP-9978 Rev. 1
		RELEASE DATE: October 1988



UNISYS

**System 80
OS/3**

**Consolidated Data
Management**

**Programming
Guide**

OS/3 Release 12.0

October 1988

Priced Item

Printed in U S America
UP-9978 Rev. 1



UNISYS

**System 80
OS/3**

**Consolidated Data
Management**

**Programming
Guide**

Copyright © 1988 Unisys Corporation
All rights reserved.
Unisys is a trademark of Unisys Corporation.

OS/3 Release 12.0

October 1988

Priced Item

Printed in U S America
UP-9978 Rev. 1

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded, using the User Comments form at the back of this manual or remarks addressed directly to Unisys Corporation, to E/MSG Product Information, P.O. Box 500, M.S. E5-114, Blue Bell, PA 19424 U.S.A.

PAGE STATUS SUMMARY
ISSUE: UP-9978 Rev. 1
RELEASE LEVEL: 12.0 Forward

Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level
Cover								
Title Page/Disclaimer								
PSS	iii							
About This Guide	v thru viii							
Contents	ix thru xv							
1	1 thru 10							
2	1 thru 4							
3	1 thru 5							
4	1 thru 29							
5	1 thru 28							
6	1 thru 7							
7	1, 2							
Appendix A	1 thru 8							
Appendix B	1 thru 9							
Appendix C	1 thru 14							
Appendix D	1, 2							
Appendix E	1							
Index	1 thru 6							
User Comments Form								
Back Cover								



About This Guide

Purpose

This guide instructs the Unisys System 80 programmer in the use of Unisys Operating System/3 (OS/3) consolidated data management.

Scope

This guide

- Explains what consolidated data management is, how it works, describes the data structures that are involved, and explains how errors and exceptions are handled.
- Describes the formats and conventions for the various types of files and discusses those things you must consider when using each type of file in your program.

Audience

The primary audience for this guide is the programmer who requires information on the various types of files that can be used in a program.

Prerequisites

Anyone using this guide should be familiar with OS/3.

How to Use This Guide

Read the entire guide to familiarize yourself with consolidated data management; then use it, as needed, to provide information on the types of files you intend to use in your program.

Organization

This guide contains seven sections and five appendixes.

Section 1. Introduction

This section describes what consolidated data management is, how it works, describes the data structures that are involved, and explains how errors and exceptions are handled.

Section 2. Card Formats and File Conventions

This section describes how card files are organized, shows the card record formats, and discusses those things you must consider when using card files in your program.

Section 3. Printer Formats and File Conventions

This section describes how printerfiles are organized, shows the printer record formats, and discusses those things you must consider when using printer files in your program.

Section 4. Magnetic Tape Formats and File Conventions

This section describes how magnetic tape files and volumes are organized, shows the tape record formats, and discusses those things you must consider when using magnetic tape files in your program.

Section 5. Disk and Format Label Diskette Formats and File Conventions

This section describes how these files are organized, shows the record formats, provides formulas for estimating space requirements for a data file, and discusses those things you must consider when using these files in your program.

Section 6. Data Set Label Diskette Formats and File Conventions

This section describes how diskette files are organized, shows the record formats, and discusses those things you must consider when using diskette files in your program.

Section 7. Workstation Formats and File Conventions

This section describes how workstation files are organized, shows the record formats, and discusses those things you must consider when using workstation files in your program.

Appendix A. Functional Characteristics of Input/Output Devices

This appendix provides summaries of the functional characteristics of the various I/O devices.

Appendix B. Code Correspondences

This appendix provides the hexadecimal value, the binary value, and the Hollerith punched card code for all ASCII and EBCDIC characters.

Appendix C. DD Job Control Statement Processing

This appendix describes how the the DD job control statement can be used to temporarily change file parameters at run time.

Appendix D. Shared Code and Accelerated File Access

This appendix describes how you use shared code to improve the performance of consolidated data management when accessing files.

Appendix E. Data Management Debugging Facility

This appendix describes how you can use the data management debugging facility to provide you with a system dump when an unexpected data management error occurs.

Related Product Information

The following Unisys documents may be useful in understanding and using consolidated data management:

Note: Throughout this guide, when we refer you to another manual, use the current version that applies to the software level in use at your site.

Interactive Services Operating Guide (UP-9972)

This guide describes the procedures used to communicate interactively through a local workstation or remote terminal with the operating system.

System/32, 34 to OS/3 Conversion Guide (UP-9318)

This guide describes the guidelines for converting from IBM System/32, 34 to OS/3.

Installation Guide (UP-8839)

This guide describes how to generate, install, and tailor your operating system.

Supervisor Technical Overview (UP-8831)

This overview describes the concepts of the component that provides the central control necessary for the optimum utilization of system hardware and software.

System Service Programs (SSP) Operating Guide (UP-8841)

This guide describes the system service programs.

Data Utilities Operating Guide (UP-8834)

This guide describes how to reproduce and maintain data files.

Job Control Language Programming Guide (UP-9986)

This guide provides information on the format and use of job control statements.

System Messages Reference Manual (UP-8076)

This handbook lists and describes the messages that are displayed on the system console during program execution.

COBOL, 1974 American National Standard Programming Reference Manual (UP-8613)

This manual describes how to use the 1974 ANSI COBOL programming language.

FORTRAN IV™ Programming Reference Manual (UP-8814)

This manual describes how to use the FORTRAN IV programming language.

Report Program Generator II (RPG II) Programming Guide (UP-8067)

This guide describes how to use the RPG II programming language.

Operations Guide (UP-8859)

This guide describes the procedures and commands used to operate System 80.

FORTRAN IV is a trademark of SuperSoft Associations.

Contents

	About This Guide	iii
Section 1.	Introduction	
	1.1. General Information	1-1
	1.2. What Is Consolidated Data Management?	1-1
	1.3. How Consolidated Data Management Works	1-3
	1.4. Data Structure	1-4
	1.5. Error and Exception Handling	1-8
	1.5.1. Error Messages	1-8
	1.5.2. Return of Control	1-8
	1.6. Related Software	1-8
	1.6.1. System Service Programs (SSP)	1-8
	1.6.2. Job Control	1-9
	1.6.3. Supervisor	1-10
	1.6.4. Data Utilities	1-10
Section 2.	Card Formats and File Conventions	
	2.1. General Information	2-1
	2.2. File Organization	2-1
	2.2.1. Card Input Files	2-2
	2.2.2. Card Output Files	2-2
	2.3. Card File Programming Considerations	2-3
Section 3.	Printer Formats and File Conventions	
	3.1. General Information	3-1
	3.2. File Organization	3-1
	3.2.1. Text	3-1
	3.2.2. Tabular Data	3-1
	3.2.3. Data on Preprinted Forms	3-2
	3.3. Printer Record Formats	3-3
	3.4. Vertical Format and Load Code Buffers	3-4
	3.5. Printer File Job Control Considerations	3-4

Section 4. Magnetic Tape Formats and File Conventions

4.1. General Information	4-1
4.2. Tape Volume and File Organization	4-1
4.2.1. EBCDIC Standard Volume Organization	4-2
4.2.2. EBCDIC Nonstandard Volume Organization	4-6
4.2.3. EBCDIC Unlabeled Volume Organization	4-9
4.2.4. ASCII Standard Volume Organizations	4-10
ASCII End-of-File and End-of-Volume Coincidence	4-14
4.2.5. Magnetic Tape File Record Formats	4-16
4.3. Magnetic Tape File Job Control Considerations	4-19
4.3.1. Assigning a Tape Device to Your Job (DVC)	4-20
4.3.2. Specifying Tape Volume Information (VOL)	4-21
Inhibiting Volume Serial Number Checking	4-21
4.3.3. Specifying Tape Label Information (LBL)	4-22
Specifying the File Identifier	4-22
Checking and Creating Volume and File Serial Numbers	4-22
Specifying the File Expiration Date	4-24
Specifying the File Creation Date	4-24
Specifying the File Sequence Number	4-24
Specifying the File Generation and Version Numbers	4-24
4.3.4. Defining Your Logical File (LFD)	4-25
4.3.5. Preparing Tape Volumes	4-26
4.3.6. Specifying Mode Characteristics for Tape Volumes	4-27
4.3.7. Creating Multivolume Files	4-27
4.3.8. Extending Tape Files	4-28

Section 5. Disk and Format Label Diskette Formats and File Conventions

5.1. General Information	5-1
5.1.1. How Disk Files Are Organized	5-1
5.1.2. Disk Access Method	5-2
MIRAM Concepts	5-2
5.2. MIRAM File Organization	5-4
5.2.1. The Data Partition	5-4
5.2.2. Entries in the Index Partition	5-8
5.2.3. MIRAM Index Structure	5-9
5.2.4. Estimating Disk Space Required for an Indexed Disk File	5-10
5.2.5. Estimating Disk Space Requirements for a Nonindexed MIRAM File	5-15
5.2.6. Minimum Cylinder and Track Allocation for MIRAM Files	5-15
5.3. Disk File Job Control Considerations	5-16
5.3.1. Device Assignment Set for Creating a Disk File	5-16
5.3.2. Device Assignment Set for Allocating Fixed-Head Area to a File on the 8417 Disk	5-17
5.3.3. Device Assignment Set for Creating a Format Label Diskette File Using the Autoloader Feature of the 8420 Diskette	5-19
5.3.4. Device Assignment Set for an Existing Disk File	5-21

5.3.5. Extending an Existing Disk File	5-21
5.3.6. Device Assignment Set for a Remote Disk File	5-21
5.3.7. Preparing Disk Volumes	5-22
5.4. Disk File Sharing	5-22
5.4.1. Logical Access Path (LAP)	5-23
5.4.2. Share Requirements	5-23
5.4.3. ACCESS Parameter Specifications	5-21

Section 6. Data Set Label Diskette Formats and File Conventions

6.1. General Information	6-1
6.2. Data Set Label Diskette File Organization	6-1
6.2.1. File Layout and Record Formats for Data Set Label Diskette Files	6-3
6.3. Data Set Label Diskette File Job Control Considerations	6-4
6.3.1. Device Assignment Set for Creating a Data Set Label Diskette File	6-5
6.3.2. Device Assignment Set for Creating a Data Set Label Diskette File Using the Autoloader Feature of the 8420 Diskette	6-6
6.3.3. Device Assignment Set for an Existing Data Set Label Diskette File	6-7
6.3.4. Preparing a Data Set Label Diskette Volume	6-7

Section 7. Workstation Formats and File Conventions

7.1. General Information	7-1
7.2. File Organization	7-1
7.3. Workstation Record Formats	7-1
7.4. Workstation File Job Control Considerations	7-1
7.4.1. Device Assignment Set for a Single-Volume Workstation File	7-2
7.4.2. Device Assignment Set for a Multivolume Workstation File	7-2

Appendix A. Functional Characteristics of Input/Output Devices

Appendix B. Code Correspondences

B.1. General Information	B-1
B.2. EBCDIC/ASCII/Hollerith Correspondence	B-1
B.2.1. Hollerith Punched Card Code	B-2
B.2.2. EBCDIC	B-2
B.2.3. ASCII	B-2
B.3. Other Card Codes	B-8
B.3.1. Compressed Card Code	B-8
B.3.2. Column Binary (Image) Code	B-9

Appendix C. DD Job Control Statement Processing

C.1. General Information	C-1
C.2. DD Job Control Statement Parameters	C-1
C.2.1. Record Format (RCFM)	C-2
C.2.2. Data Buffer/Block Size (BKSZ)	C-3
C.2.3. Record Size (RCSZ)	C-5
C.2.4. Key Length (KLENN)	C-5
C.2.5. Key Location (KLOCn)	C-5
C.2.6. Index Buffer Size (INDS)	C-5
C.2.7. Initial Space Allocation Percentages (SIZE, SIZE1, SIZE2)	C-6
C.2.8. File Sharing Characteristics (ACCESS)	C-6
C.2.9. Variable Sector Support (VSEC)	C-7
C.2.10. File Recovery Support (RECV)	C-7
C.2.11. One Volume Online at a Time (VMNT)	C-10
C.2.12. Record Control Byte (RCB)	C-10
C.2.13. Offset Physical Sector (OFFSET)	C-11
C.2.14. General Rewind Options (REWIND)	C-11
C.2.15. Rewinding at File Open (OPRW)	C-11
C.2.16. Rewinding at File Close (CLRW)	C-11
C.2.17. File Label Type (FILABL)	C-12
C.2.18. Tape Marks (TPMARK)	C-12
C.2.19. Restoring Initialized Files (RESTORE)	C-12
C.2.20. Disk Cache Support (CACHE)	C-13
C.2.21. Suppressing Error Messages (MSGSUPP)	C-14

Appendix D. Shared Code and Accelerated File Access

Appendix E. Data Management Debugging Facility

E.1. General Information	E-1
E.2. Console Command	E-5

Index

User Comments Form

Figures

1-1.	Relationship of Consolidated Data Management to a Program	1-2
1-2.	Consolidated Data Management Program Development Cycle	1-3
1-3.	Organization of Data on Peripheral Devices	1-5
2-1.	Typical Card File Structure	2-1
2-2.	Fixed-Length, Unblocked Record Format for Card Input Files	2-2
2-3.	Record Formats for Card Output Files	2-3
3-1.	Sample Tabular Data	3-2
3-2.	Sample of Data on Preprinted Form	3-2
3-3.	Printer Record Formats	3-3
4-1.	Organization for a Standard Labeled EBCDIC Magnetic Tape Volume - Single File	4-3
4-2.	Organization for a Standard Labeled EBCDIC Magnetic Tape Volume - Multifile Volume with End-of-File Condition	4-4
4-3.	Organization for a Standard Labeled EBCDIC Magnetic Tape Volume - Multifile Volume with End-of-Volume Condition	4-5
4-4.	Organization for a Nonstandard EBCDIC Magnetic Tape Volume - Single File	4-7
4-5.	Organization for a Nonstandard EBCDIC Magnetic Tape Multifile Volume	4-8
4-6.	Organization for an Unlabeled EBCDIC Magnetic Tape Volume	4-9
4-7.	Organization of ASCII Single-File, Single-Volume and Single-File, Multivolume Sets	4-11
4-8.	Volume Organization of an ASCII Multifile, Single-Volume Set	4-12
4-9.	Volume Organization of an ASCII Multifile, Multivolume Set	4-13
4-10.	Label Configuration Options of an ASCII Multifile, Multivolume Set when End-of-Volume and End-of-File Coincide	4-15
4-11.	Record and Block Formats for Magnetic Tape Files, ASCII and EBCDIC	4-16
5-1.	Disk (MIRAM) Data Record Slots Spanning Physical Sector Boundaries	5-5
5-2.	Disk (MIRAM) Data Record Formats	5-6
5-3.	Fine-Level Index Block	5-8
5-4.	Coarse- or Mid-Level Index Block	5-9
6-1.	Data Set Label Diskette File Layout	6-3
6-2.	Data Set Label Diskette Record Formats	6-4
B-1.	Compressed Card Code	B-8
B-2.	Column Binary (Image) Card Code	B-9



Tables

4-1.	Effects of Job Control VOL and LBL Statements when a File Is Opened, Standard Labeled Tape File	4-23
5-1.	Disk-Dependent Factors for Estimating Disk Space Requirements	5-12
5-2.	Minimum Cylinder and Track Allocation for MIRAM Files	5-15
5-3.	Summary of ACCESS Parameter Specifications	5-28
6-1.	Data Set Label Diskette Characteristics	6-2
A-1.	Card Reader Subsystem Characteristics	A-1
A-2.	Card Punch Subsystem Characteristics	A-3
A-3.	Printer Subsystem Characteristics	A-4
A-4.	Disk and Diskette Subsystem Characteristics	A-7
A-5.	Magnetic Tape Subsystem Characteristics	A-8
A-6.	Workstation Subsystem Characteristics	A-9
B-1.	Cross-Reference Table: EBCDIC/ASCII/Hollerith	B-3
C-1.	Allowable Keyword Parameters for the DD Job Control Statement	C-1



Section 1

Introduction

1.1. General Information

As you know, all computer programs process data in one form or another; however, the data and the program are in two different places. The program is executed in the main storage section of the central processing unit (CPU) and the data is contained on devices external to the CPU. In order to process the data and produce the desired results, data must be moved in from and out to these peripheral devices. Because the physical and electronic characteristics of the various devices differ, it can lead to problems if you have to take the characteristics of a device into consideration each time you want to perform an input/output (I/O) operation. Obviously, there is a need for some way to specify an I/O operation in a program on a logical level. The answer to this need is consolidated data management.

1.2. What Is Consolidated Data Management?

Consolidated data management is a collection of program modules that are written for each of the I/O devices supported by your system. These modules handle the actual movement of data. They take care of all the device characteristic requirements; consequently, you need not worry about this when you want to perform I/O operations. All you need to do is make a formal request in your program to data management and it moves the data in from or out to the particular I/O device. Figure 1-1 illustrates the relationship between data management and your program.

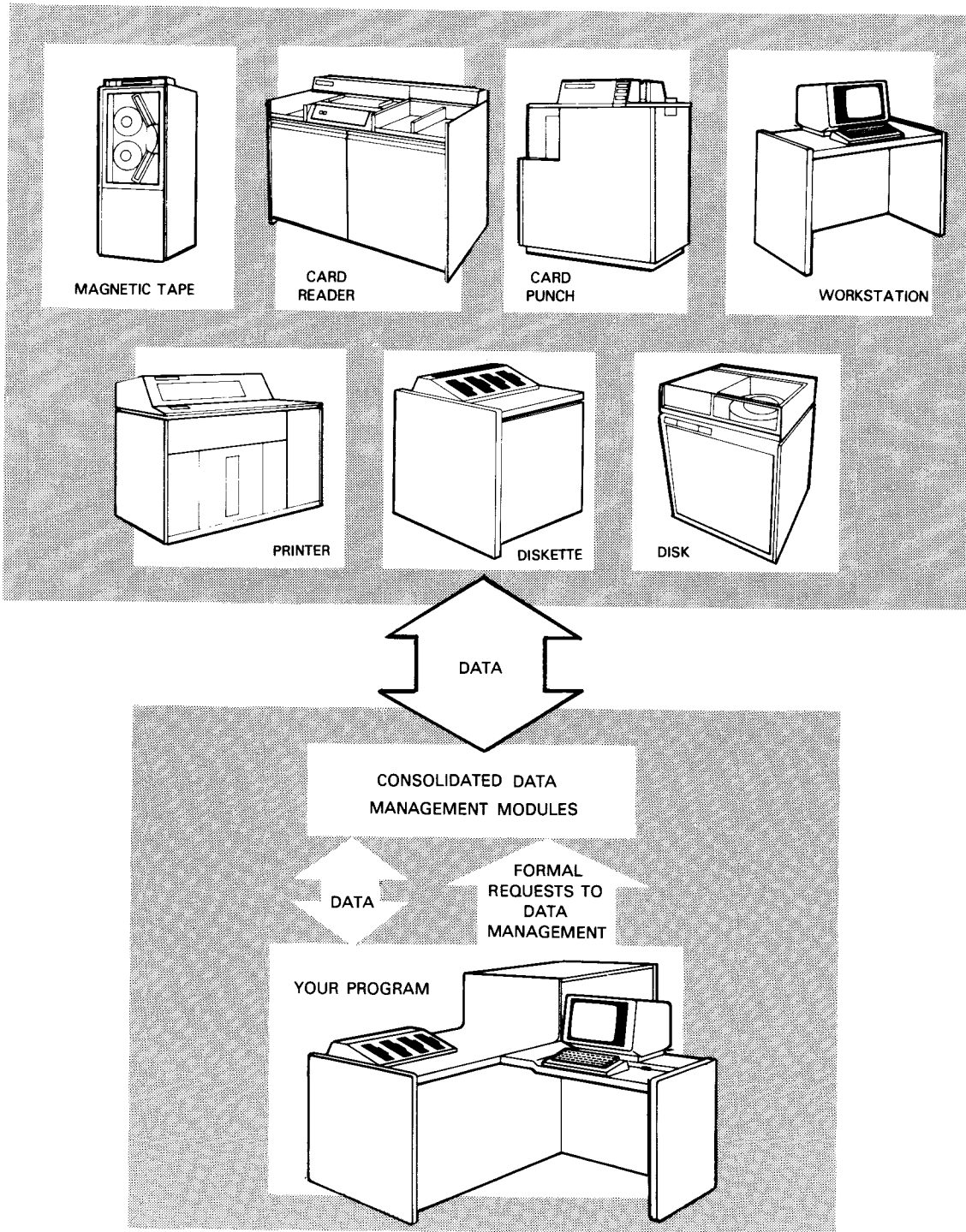


Figure 1-1. Relationship of Consolidated Data Management to a Program

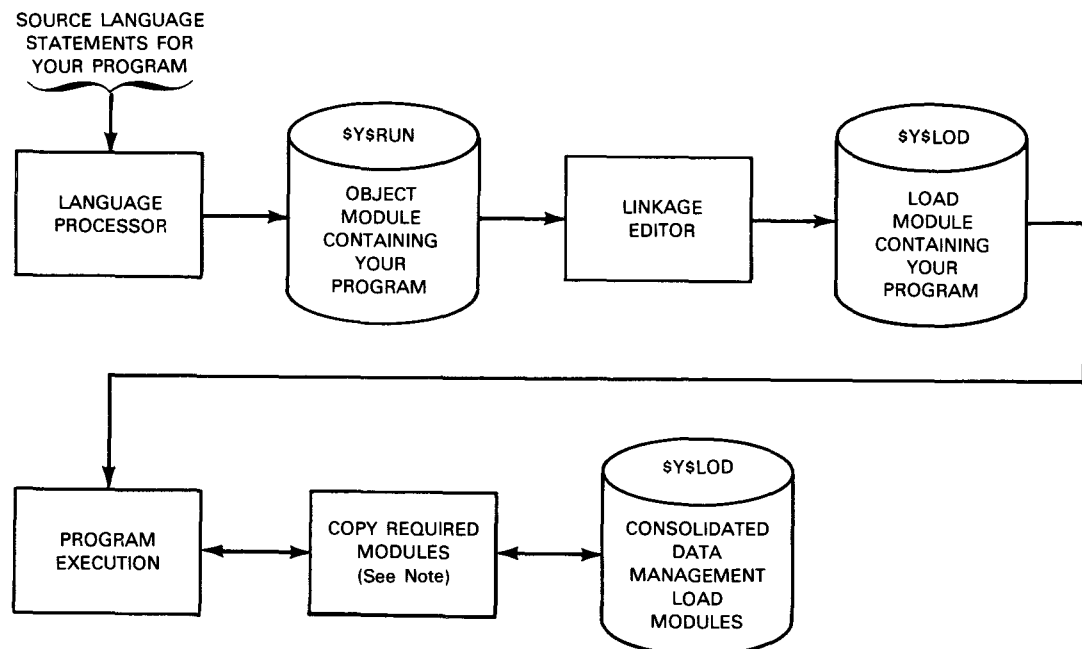
As you can see in Figure 1-1, data management acts as the data transfer mechanism between your program and the I/O devices.

1.3. How Consolidated Data Management Works

When you write your program, you establish a unique file name for each I/O device you intend to use. You then describe the characteristics for each file. Once this is done, you use these file names in conjunction with I/O commands at each point that you want to move data into or out of your program. The I/O commands act as formal requests to data management. After your program is written, it must be processed by the applicable language processor (assembler, FORTRAN, COBOL, or RPG II) to form an object module. The next step is to link edit your program to produce a load module (an executable program).

When the time comes to execute your program, you use job control language statements to associate each file name in your program with the particular device you want to get data from or send data to. Once this is done, you can proceed to execute your program. The appropriate data management modules are placed in main storage at this time as shown in Figure 1-2.

Thereafter, each time an input or output command is encountered during processing, the applicable data management module gets data from or sends data to the appropriate device.



Note: Based on device assignment sets for files in the job control stream, the required modules are copied at execution time.

Figure 1-2. Consolidated Data Management Program Development Cycle

1.4. Data Structure

Consolidated data management recognizes the following as structural entities:

- Volume

The largest physical unit for data storage; such as tape reel or disk pack.

- File

A delimited storage space having an identifying file name and consisting of a collection of related data.

- Block

That portion of a file that is transferred into or out of main storage by a single access.

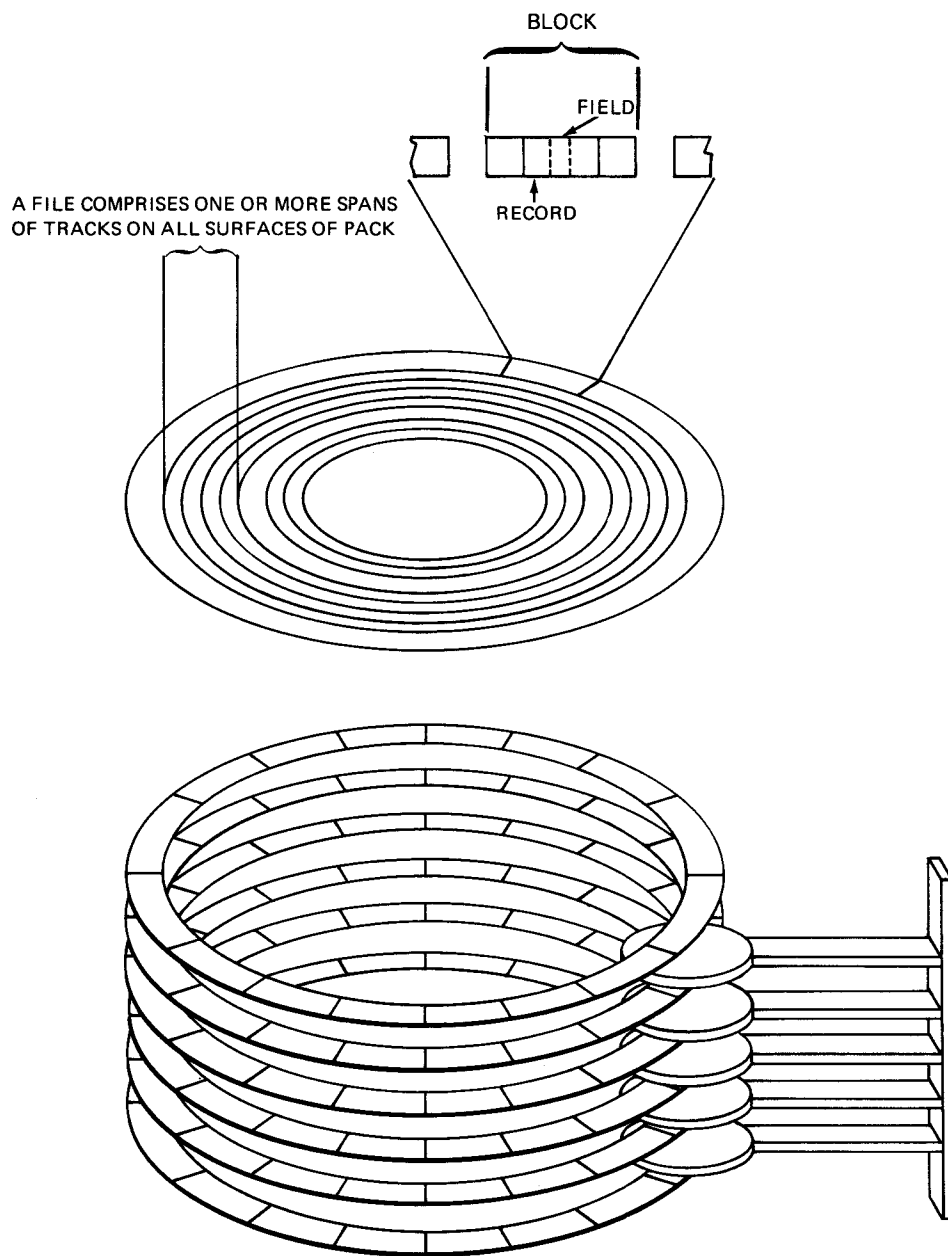
- Record

A collection of contiguous characters that you have designated to be handled as a unit.

- Field

One or more contiguous characters within a record that represents a single piece of information.

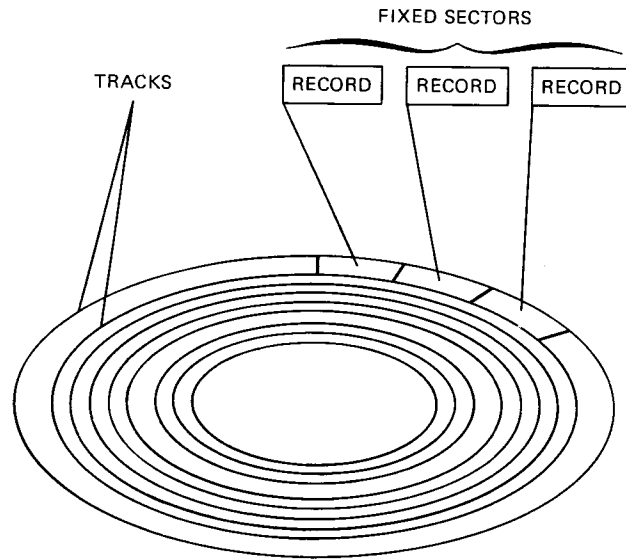
The volume concept is not truly applicable to printers, workstation, or card devices. On disk, diskettes, and magnetic tape, a file may be larger than a volume; that is, a file may require more than one physical unit to hold it. In this case, you have what is called a multivolume file. Figure 1-3 shows the organization of data on the peripheral devices supported by consolidated data management.



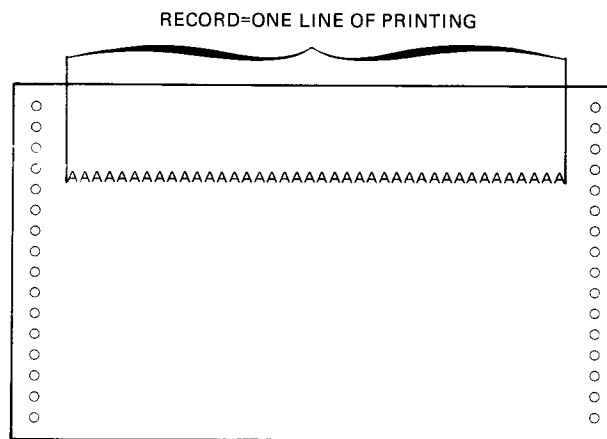
a. Disk pack

Note: The set of tracks at a specific radius on all recording surfaces is called a cylinder.

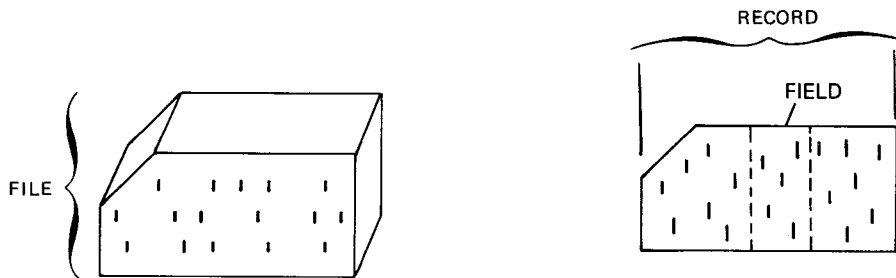
Figure 1-3. Organization of Data on Peripheral Devices (Part 1 of 3)



b. Diskette

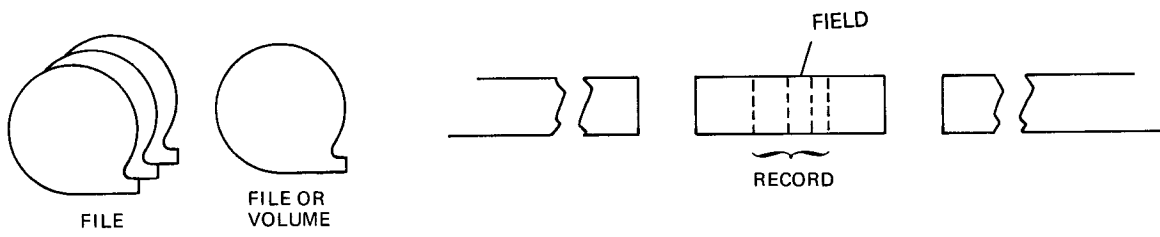


c. Printer

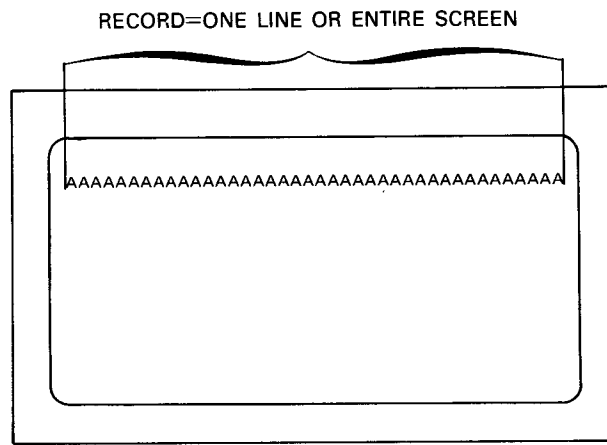


d. Punched card

Figure 1-3. Organization of Data on Peripheral Devices (Part 2 of 3)



e. Magnetic tape



f. Workstation

Figure 1-3. Organization of Data on Peripheral Devices (Part 3 of 3)

1.5. Error and Exception Handling

During program execution, consolidated data management monitors each input/output request and notifies you when an error or exception occurs so that you can take appropriate action. This notification consists of messages that are logged on the printer or displayed at the console.

1.5.1. Error Messages

The error messages issued by data management have the following format:

```
DM nn chan/device explanatory-text,TYPE=nn
```

As you can see, these messages consist of the prefix DM followed by a number, the channel and device address, and text explaining what caused the error. Some messages have the suffix TYPE=nn, where nn is a subcode that further defines the error. A complete list of the data management messages along with suggested actions is provided in the *System Messages Reference Manual* (UP-8076).

1.5.2. Return of Control

Consolidated data management is designed so that control is always returned to your program when an error is detected; that is, your program is never terminated. When an error occurs, the applicable error message is logged or displayed and control is returned to your program inline at the next sequential instruction after the instruction that caused the error.

1.6. Related Software

There are several software components that are indirectly involved with data management or perform functions required for program operation. These components include:

- System service programs (SSP)
- Job control
- Supervisor
- Data utilities

1.6.1. System Service Programs (SSP)

The SSP are those routines that you use to prepare disk, diskette, and magnetic tapes to accept data records, to create program load modules, to obtain printouts (dumps) of

main storage, and to construct and reorganize your program libraries. These routines are described in detail in the *System Service Programs (SSP) Operating Guide* (UP-8841).

The disk prep routine performs a surface analysis of the disk or diskette tracks and assigns alternate tracks if defects are discovered. It also establishes a volume table of contents (VTOC) for the device so that files may be placed on it.

The magnetic tape prep routine prepares a magnetic tape by writing the initial volume label, dummy file header label, dummy file trailer label, and tape marks.

The linkage editor is used to create your program load module.

The dump routines allow you to obtain narrative or nonnarrative printouts of main storage.

The system librarian allows you to construct and reorganize your program libraries.

1.6.2. Job Control

A major function of job control is the assignment of the required peripheral devices. These are assigned through job control statements that specify logical unit numbers, alternate device types, and information about the file. These statements include:

- DVC statement - assigns device number.
- VOL statement - identifies tape, disk, and diskette volumes.
- EXT statement - provides disk extent information (amount of disk or diskette space required for your file).
- LBL statement - identifies the physical file on tape, diskette, or disk.
- LFD statement - associates the file defined in your program with the physical file on the device.

Other functions provided by job control include loading the vertical format buffer (VFB) and the load code buffer (LCB). For details on job control, see the *Job Control Language Programming Guide* (UP-9986).

1.6.3. Supervisor

The supervisor provides the largest amount of support for your program and data management. For the most part this support is automatic and therefore is not apparent during the execution of your program. This support includes

- Physical input/output control system (PIOCS)
Handles the actual transfer of physical records between the I/O devices and main storage.
- Transient scheduling routines
Retrieve transient routines, such as the file open and close routines, and bring them into main storage for execution.
- Operator communications routines
Handle the communications concerning volume mounting requests, and so on.
- File protection routines
Protect files and records during shared file processing.
- Timer services routine
Compute run time, scheduling, and so on.
- Disk space management routines
Allocate disk space and maintain space accounting through procedures that update the volume table of contents (VTOC) on disk files.

1.6.4. Data Utilities

A data utility routine is provided to assist you in manipulating data files and preparing card decks. This routine edits or corrects data records, or copies them from one device to another.

See the *Data Utilities Operating Guide* (UP-8834) for details concerning this routine.

Section 2

Card Formats and File Conventions

2.1. General Information

A punched card file consists of a card deck that is input via a card reader or output via a card punch. Records can comprise either a portion of a card or a complete card. The basic punched cards for the card subsystems are the standard 80-column cards. However, optional hardware features allow you to read 51- or 66-column cards.

Refer to Appendix A for the functional characteristics of the card subsystems that are supported.

2.2. File Organization

Punched card files are sequential files; that is, the records are handled one at a time in sequential order. The card deck consists of a start-of-data job control card (optional), data cards containing one record each, and an end-of-data job control card.

Figure 2-1 shows a typical card file structure.

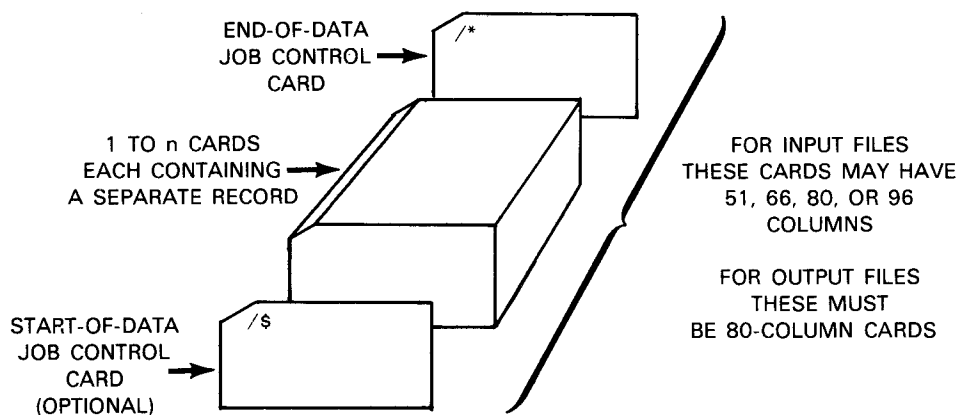
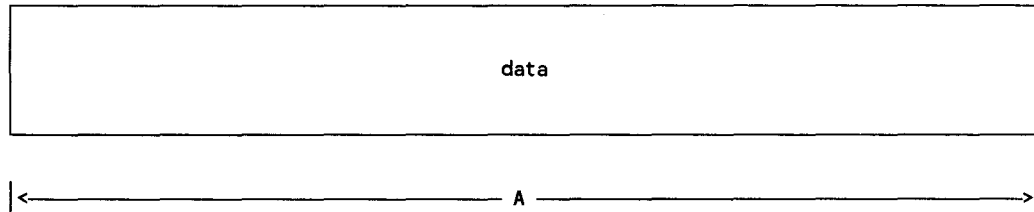


Figure 2-1. Typical Card File Structure

2.2.1. Card Input Files

A card input file consists of fixed-length, unblocked records; that is, all records are the same size. When a record is read, it is placed in the I/O area. Figure 2-2 shows the record format for fixed-length, unblocked records.



Legend

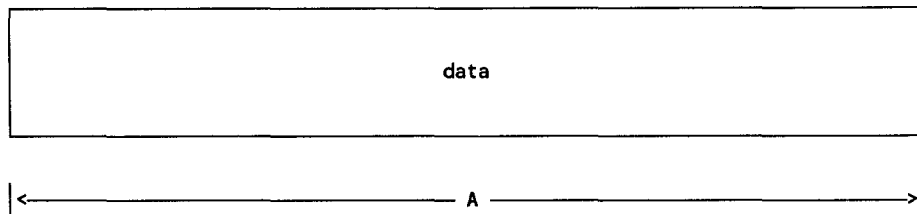
- A Data record length. The I/O area must be at least the same size as the record length and must be an even number of bytes. If you are using 51-column cards, the I/O area must be specified as 52 bytes.

Figure 2-2. Fixed-Length, Unblocked Record Format for Card Input Files

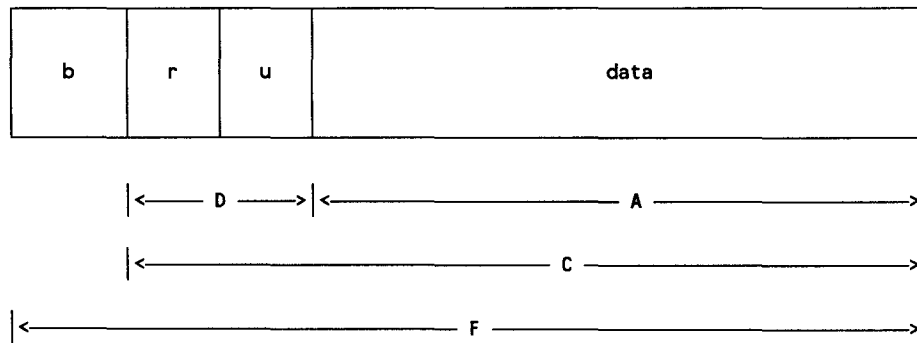
2.2.2. Card Output Files

A card output file consists of data that is formed into records, either in the I/O area or a designated work area, and then is sent to the card punch, where the records are punched in the standard 80-column card format. The output records can be fixed length or variable length. These record formats are shown in Figure 2-3.

FIXED LENGTH



VARIABLE LENGTH



Legend

- b Block size field, 4 bytes
- r Record length field, 2 bytes
- u Reserved (2 bytes); may be any 2 characters chosen by the user
- A Data record length
- C Variable record length
- D Record size field
- F I/O area layout. The I/O area must be an even number of bytes and its size must equal the maximum record size plus the block size and record size fields if you are dealing with variable-length records.

Figure 2-3. Record Formats for Card Output Files

2.3. Card File Programming Considerations

The following points must be considered when you use card files with your program:

1. The file name that you use in your program for a card file must be associated with a card I/O device via a DVC-LFD job control statement series in the job control stream at program execution. See the job control user guide for details on these statements.
2. If you intend to process 51- or 66-column cards, the device number that you specify in the DVC job control statement for that file must be the number of a device that supports the required feature.

Card Formats and File Conventions

3. If your files require either input or output translation (the conversion of an existing character to another), you must provide a translation table in your program. See the applicable programming language guide for details.
4. If you intend to read or write ASCII files, you must indicate this when you describe the file in your program. This is necessary because the internal code in the system is EBCDIC and data management must be alerted so that it can translate the incoming data to EBCDIC (via the data management ASCII-to-EBCDIC translation table) or the outgoing data to ASCII (via data management EBCDIC-to-ASCII translation table). See the applicable programming language guide for details.

Section 3

Printer Formats and File Conventions

3.1. General Information

A printer file consists of data that you create in your program and cause to be printed, one line at a time, on a printer device. Each printed line can have up to 160 characters, depending upon the printer subsystem you use.

Refer to Appendix A for the functional characteristics of the printer subsystems that are supported.

3.2. File Organization

Printer files can be organized to produce text, tabular data, or data on preprinted forms. In each case, you must set up the data you want printed on each line, must control the vertical separation between lines, and must provide for skipping to the next page when the space on the current page is exhausted. (See the applicable programming language guide for details.)

3.2.1. Text

The simplest printer file is one that consists entirely of text. An example of this is the lines you are presently reading. If you had to write a program to produce these lines, each line would be a record and you would form each line in an I/O area or work area. Then you would cause it to be printed. This process is repeated for each line until the end of the page is reached, at which point you issue an instruction to skip to the top of the next page.

3.2.2. Tabular Data

The records for tabular data and reports are formed in the same manner as in text files (in an I/O area or work area). In these cases, your program is more complex because of vertical and horizontal spacing requirements, page and column headings, and other repetitive items (see Figure 3-1).

PAGE HEADING

COLUMN HEADINGS		DAILY ACTIVITY REPORT				DEPARTMENT
PART	ITEM	TRANS-	QUAN	REOR		
NUMBER	DESCRIPTION	ACTION	ON-HAND	PO	BILLED	
00010E	CAPACITOR	ORDER			PRODUCTION	
00010F	ROTOR	ORDER			PRODUCTION	
00010G	POINT, IGN	ORDER			MAINTENANCE	

Figure 3-1. Sample Tabular Data

3.2.3. Data on Preprinted Forms

A printer file that places data on a preprinted form is easy to use once it is organized. You form your records in an I/O area or work area as with text or tabular data. The difference, as you can see in Figure 3-2, is that you have to closely control the positioning of your data.

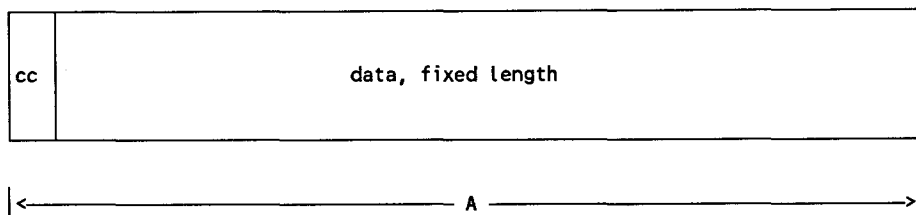
UNISYS					
P. O. BOX 500 BLUE BELL, PA. 19422					
			INT		UMS
SITE 3-1					
ATTN: CATHY SMITH					
D6866M	8598	UP	8071		00851
ADDRESS CORRECTION REQUESTED RETURN POSTAGE GUARANTEED					
UD1-527					

Figure 3-2. Sample of Data on Preprinted Form

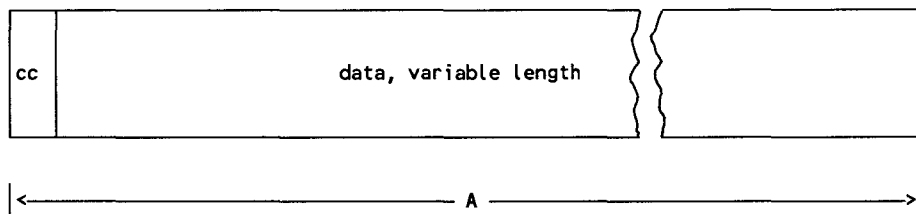
3.3. Printer Record Formats

The printer record formats are shown in Figure 3-3. As you can see, a record may contain a control character that specifies line spacing or skipping when the file is printed. This character is not printed, but is a part of the record in storage.

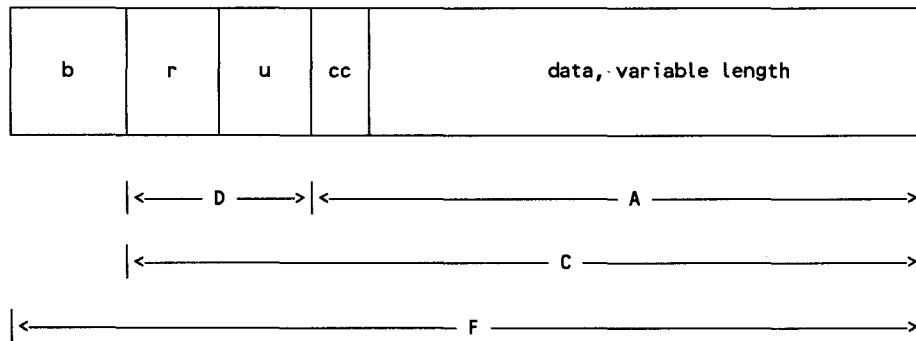
FIXED LENGTH



UNDEFINED



VARIABLE LENGTH



Legend

- b Block size field, 4 bytes
- cc Control character, 1 byte, optional
- r Record length field, 2 bytes, binary
- u Reserved (2 bytes); can be any 2 characters you choose
- A Data record length
- C Variable record length
- D Record size field
- F I/O area layout

Figure 3-3. Printer Record Formats

3.4. Vertical Format and Load Code Buffers

All printers contain a vertical format buffer and a load code buffer. The vertical format buffer is used to define the printer page in terms of the number of lines per page, the density of the lines, the overflow line, and the codes you can use with your program instructions to skip to specific lines on a page.

The load code buffer specifies the 8-bit codes that are associated with the graphic symbols on the print cartridge, band, or drum.

Normally, the vertical format buffer and the load code buffer are set up at system generation time for each printer type. For details, refer to the *Installation Guide* (UP-8839).

Normally, your only concern with these buffers is to know what standards are established for the printer you are going to use with your program. In most cases, these standards allow you to produce the printed output you require.

3.5. Printer File Job Control Considerations

The following points must be considered when you use printer files with your program:

1. The file name that you use in your program for a printer file must be associated with a printer device via a DVC-LFD job control statement series in the job control stream at program execution. See the *Job Control Language Programming Guide* (UP-9986) for details on these statements.
2. If you intend to process records that are larger than 120 characters, the device number that you specify in the DVC job control statement for that file must be the number of a device that has the required feature.
3. You must include a VFB job control statement in the DVC-LFD statement series for your printer file if:
 - You want to use one of the filed vertical format buffers (established at SYSGEN time or by use of the job SG\$PRB) rather than the default vertical format buffer used when no VFB statement is specified.
 - Your forms control requirements cannot be met by the vertical format buffer that was established at SYSGEN time. In this case, the VFB statement is used to specify a unique vertical format that meets your needs. When the program is executed, the vertical format buffer you specified via the VFB statement is used instead of the one specified at SYSGEN time. See the *Job Control Language Programming Guide* (UP-9986) for details on the VFB statement.

4. You must include an LCB job control statement in the DVC-LFD statement series for your printer file if
 - You want to use one of the filed load code buffers (established at SYSGEN time or by use of the job SG\$PRB) rather than the default load code buffer used when no LCB statement is specified.
 - Your requirements cannot be met by the load code buffer that was established at SYSGEN time. The LCB statement is used to specify a unique load code buffer that meets your needs. When the program is executed, the load code buffer you specified via the LCB statement is used instead of the one specified at SYSGEN time.

See the *Job Control Language Programming Guide* (UP-9986) for details on the LCB and VFB statements.



Section 4

Magnetic Tape Formats and File Conventions

4.1. General Information

Magnetic tape files consist of data records that are recorded on one or more volumes (reels) of magnetic tape. These files are sequential files; the data records are recorded on tape in the order in which they are submitted, and they are read from the tape starting with the first record on tape and continuing with each successive record. The recording and reading are accomplished via a tape subsystem. Refer to Appendix A for the functional characteristics of the magnetic tape subsystems that are supported.

4.2. Tape Volume and File Organization

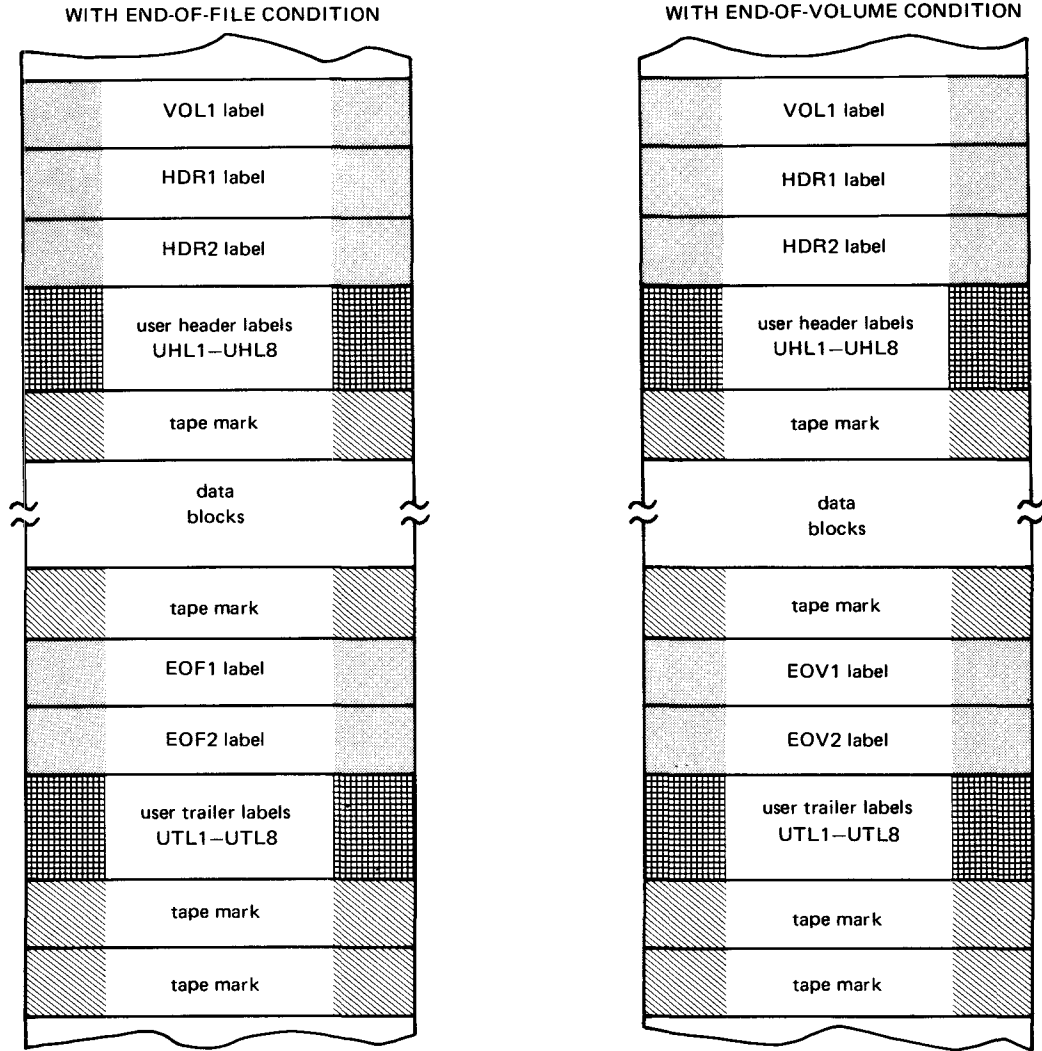
Consolidated data management allows you to process magnetic tape files encoded in Extended Binary-Coded Decimal Interchange Code (EBCDIC) as well as in the *American National Standard Code for Information Interchange (ASCII), X3.4-1977*. The file structure, organization, and processing specifications for ASCII magnetic tape files is described in the *American National Standard Magnetic Tape Labels for Information Interchange, X3.27-1969*. Both of these standards are followed when ASCII magnetic tape files are processed. The tape volumes (reels) can be organized as follows:

- EBCDIC
 - Standard labeled
 - Nonstandard labeled
 - Unlabeled
- ASCII
 - Standard labeled

The paragraphs that follow explain and illustrate the different types of volume organization.

4.2.1. EBCDIC Standard Volume Organization

A standard volume has system standard labels and required tape marks; it may also, at your option, contain standard user header labels and user trailer labels (UHL and UTL). All standard tape labels are written in blocks of 80 bytes. Data management assumes that the labels appear on tape in the order shown in Figures 4-1, 4-2, and 4-3, which illustrate the reel organization for standard labeled EBCDIC volumes with an end-of-file (EOF) and an end-of-volume (EOV) condition. A standard labeled EBCDIC volume processed by data management ends in either an EOF or an EOV label group, followed by two tape marks. The second tape mark indicates that no valid information follows. No provision is made for creating additional volume, header, or EOF/EOV labels on output files; if they exist on input files, data management bypasses them.



Legend





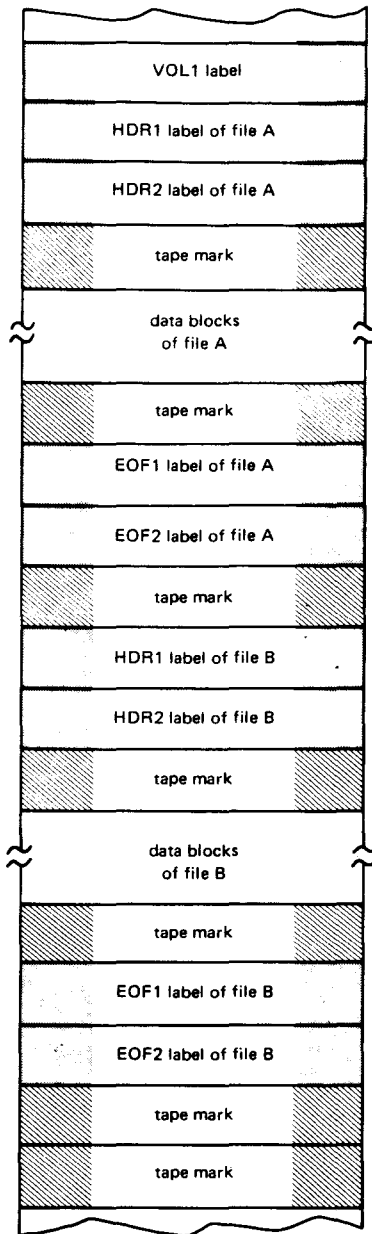
-  Content supplied by user.
-  These bytes are required and generated by data management.
-  These bytes are generated by data management; user supplies content for certain fields.
-  These bytes are generated by user's routine for processing these labels; content is at user's option except for content of 4-byte label-id fields. User is limited to eight UHL and eight UTL.

Figure 4-1. Organization for a Standard Labeled EBCDIC Magnetic Tape Volume - Single File

Magnetic Tape Formats and File Conventions

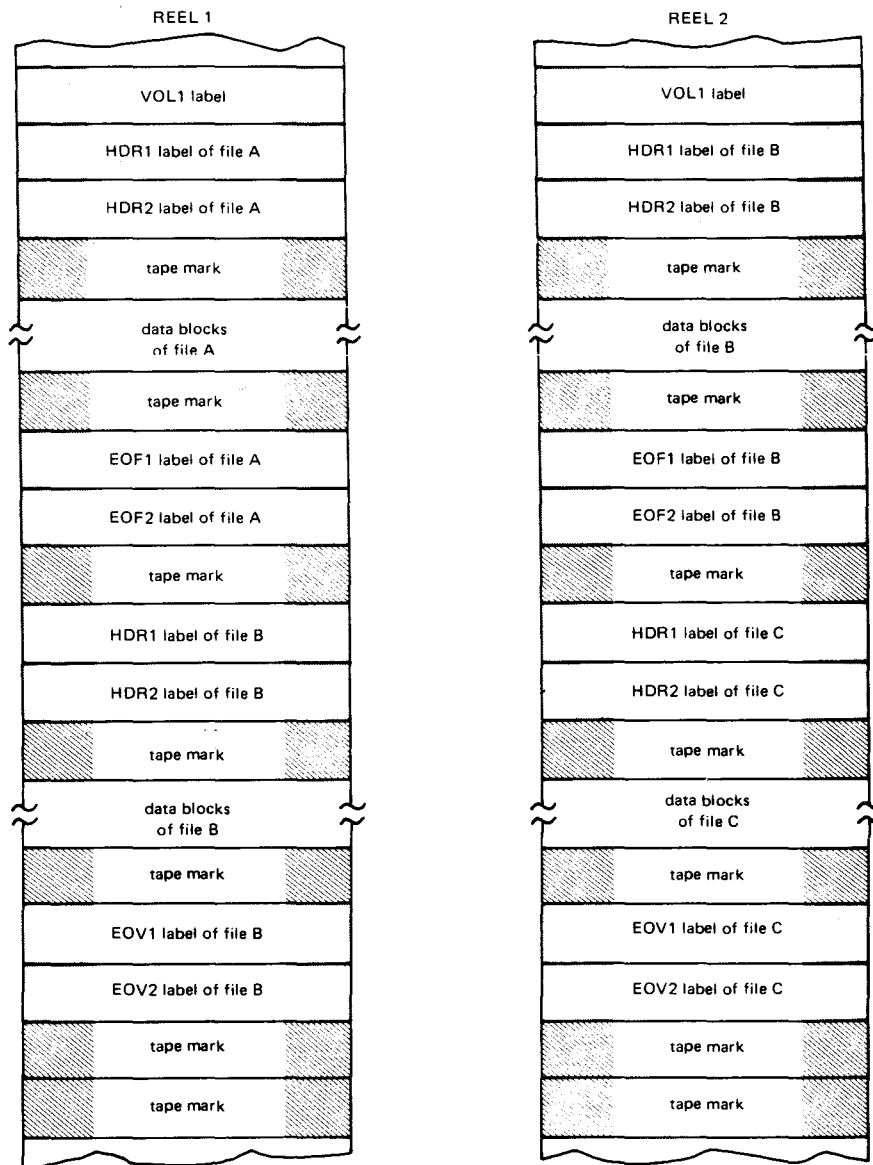


Legend

- Content supplied by user.
- These bytes are required and generated by data management.
- These bytes are generated by data management; user supplies content for certain fields.

Note: Assume that file B completes on this volume.

Figure 4-2. Organization for a Standard Labeled EBCDIC Magnetic Tape Volume - Multifile Volume with End-of-File Condition



Legend

- Content supplied by user.
- These bytes are required and generated by data management.
- These bytes are generated by data management; user supplies content for certain fields.

Note: Assume that file C is not completed on reel 2, but carries over (like file B) onto another volume. If file C was completed on reel 2, its EOV1 and EOV2 labels shown here would be replaced with EOF1 and EOF2 labels.

Figure 4-3. Organization for a Standard Labeled EBCDIC Magnetic Tape Volume - Multifile Volume with End-of-Volume Condition

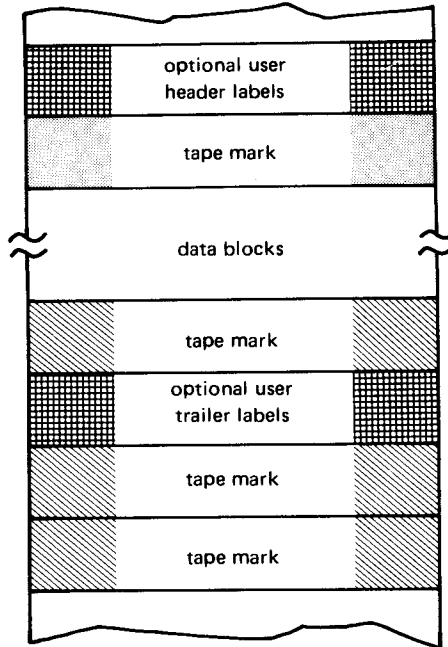
4.2.2. EBCDIC Nonstandard Volume Organization

A nonstandard volume is any volume that contains only nonstandard labels and certain required tape marks. Figures 4-4 and 4-5 show the formats for EBCDIC nonstandard volumes.

The optional user header and trailer labels may be of any format, length, or number because they are handled by a label processing routine that you supply in your program.

The tape mark following the user header label is only required if you intend to use read-backward operations in your program or if you intend to omit label checking. It is not required and may be omitted if you intend to perform label checking. Normally, this tape mark is automatically generated by data management unless you specify otherwise.

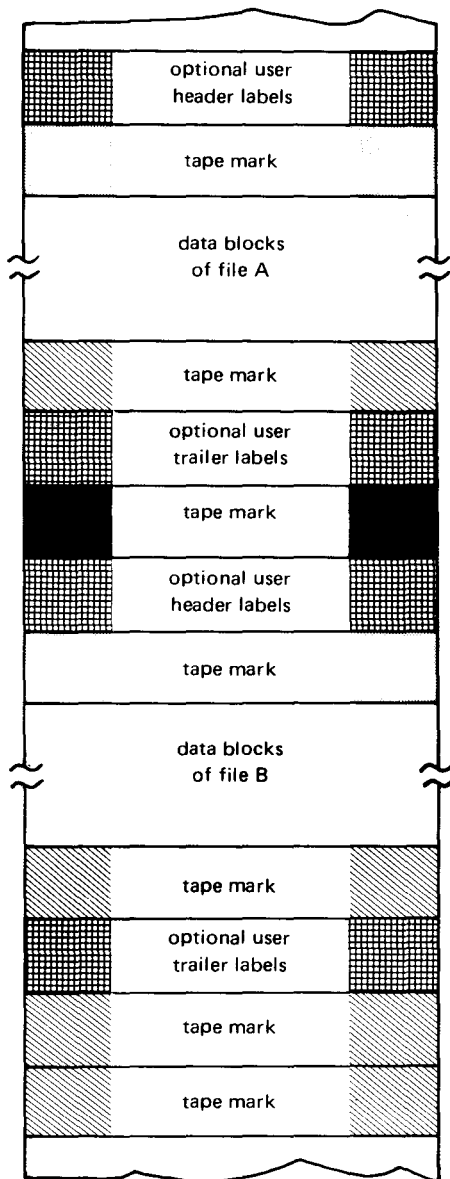
The tape mark following the data blocks is required and is automatically generated by data management. If user trailer labels are present, data management automatically generates the two required tape marks that must follow these labels. If the user trailer labels are not present, data management writes only one additional tape mark after the one following the data blocks. This second tape mark is always present when a file is the only file or is the last file on the volume. If the volume is a multifile volume, this second tape mark is overwritten by the next file placed on this volume.



Legend

- Content supplied by user.
- These bytes are required and generated by data management; only two tape marks follow data blocks if UTL are not present.
- These bytes are generated by data management unless user specifies otherwise; required only if label checking is omitted or read-backward operations are specified.
- The presence, content, format, and number of these bytes are entirely at user's option.

Figure 4-4. Organization for a Nonstandard EBCDIC Magnetic Tape Volume - Single File



Legend






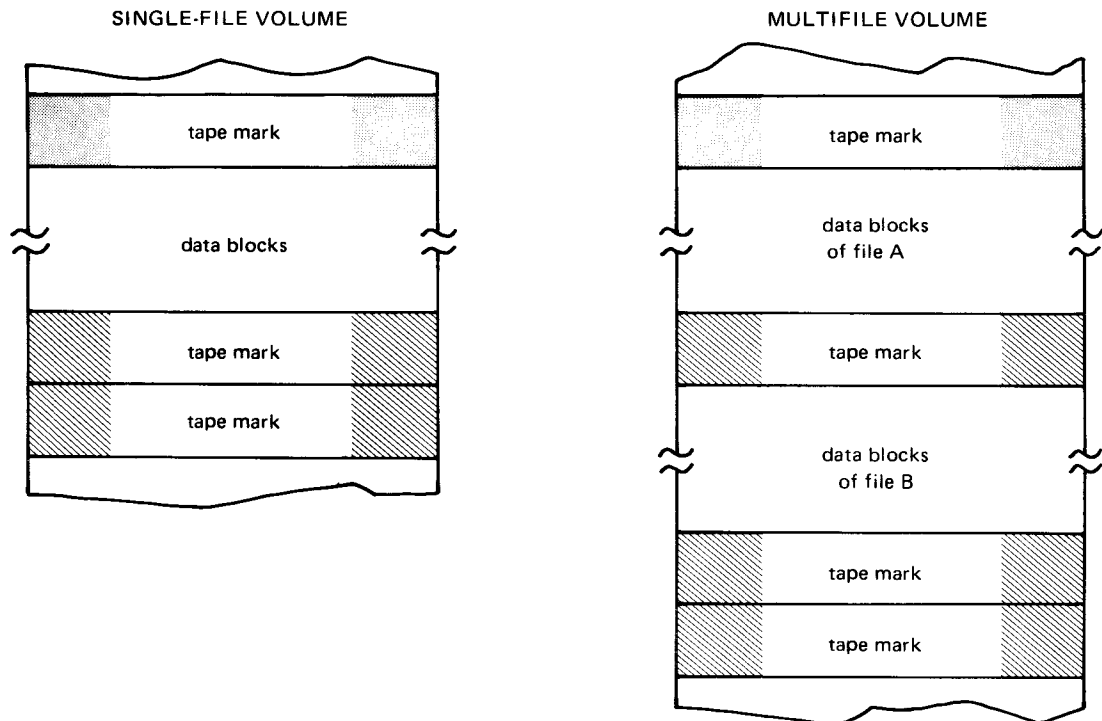
-  Content supplied by user.
-  These bytes are required and generated by data management; only two tape marks follow data blocks of last file on volume if UTL are not present.
-  These bytes are generated by data management unless user specifies otherwise; required only if label checking is omitted or read-backward operations are specified.
-  Presence, content, format, and number of these bytes are entirely at user's option.
-  Always present; written by data management.

Figure 4-5. Organization for a Nonstandard EBCDIC Magnetic Tape Multifile Volume

4.2.3. EBCDIC Unlabeled Volume Organization

Consolidated data management can also process unlabeled tape volumes. Figure 4-6 shows the organization for unlabeled EBCDIC volumes.

A tape mark normally precedes the data block unless you specify otherwise. The tape mark following the data blocks is required on both single-file and multifile volumes and is automatically generated by data management. A second tape mark is always written by data management following the last or only file on each volume. If the volume is a multifile volume, this second tape mark is overwritten by the next file placed on this volume.



Legend

- Content supplied by user.
- These bytes are required and generated by data management; two tape marks follow data blocks of last file on volume.
- These bytes are generated by data management unless user specifies otherwise; required only when user specifies read-backward operations.

Figure 4-6. Organization for an Unlabeled EBCDIC Magnetic Tape Volume

4.2.4. ASCII Standard Volume Organizations

The *American National Standard, X3.27-1978*, provides for the following file sets (collections of one or more related files recorded on one or more volumes):

- Single file, single volume
- Single file, multivolume
- Multifile, single volume
- Multifile, multivolume

These volume organizations are shown in Figures 4-7 through 4-10. Note that all ASCII tape files contain standard labels. Since data management follows the standard, it expects to find these labels on ASCII input files and it generates them for ASCII output files.

Magnetic Tape Formats and File Conventions

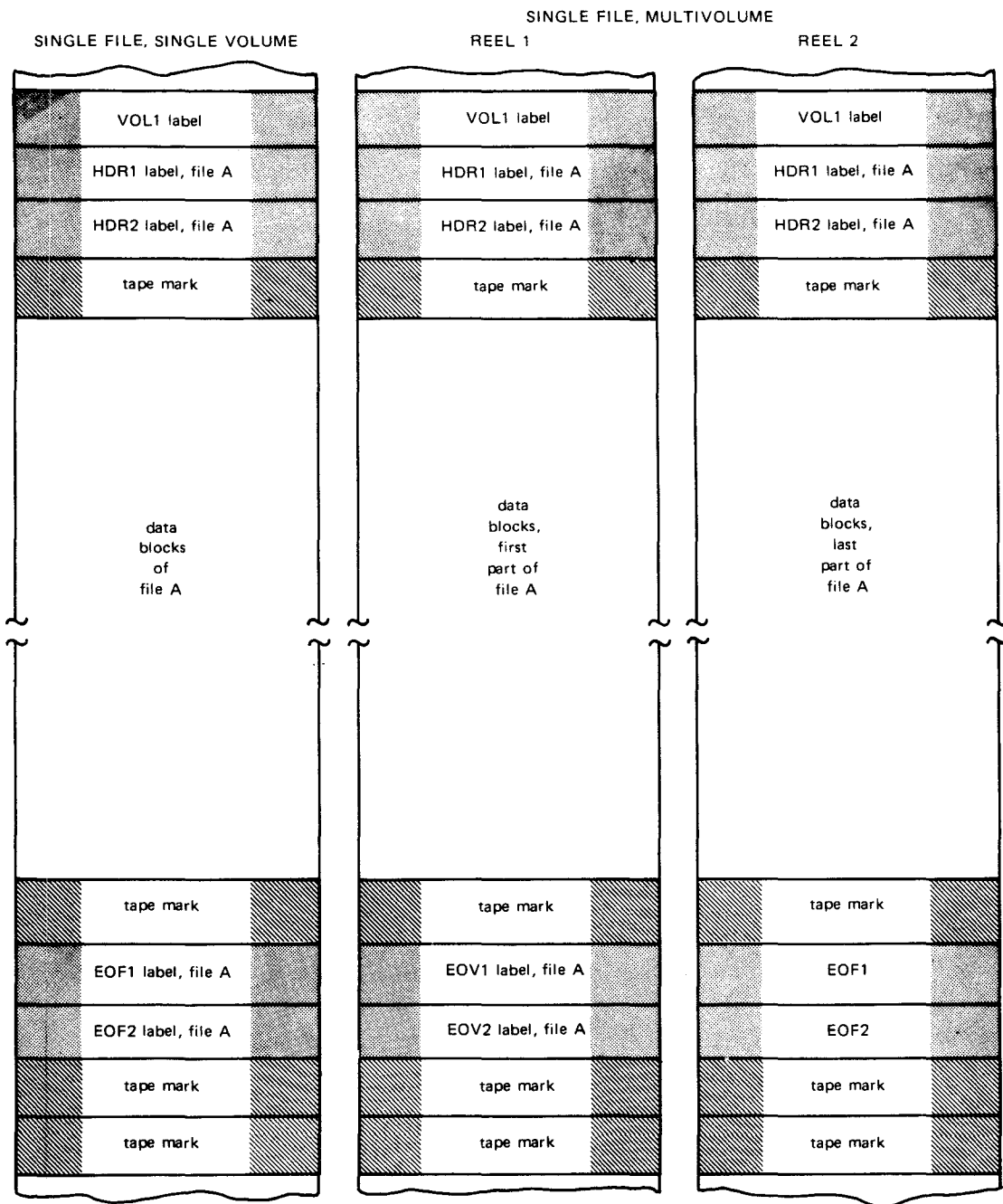
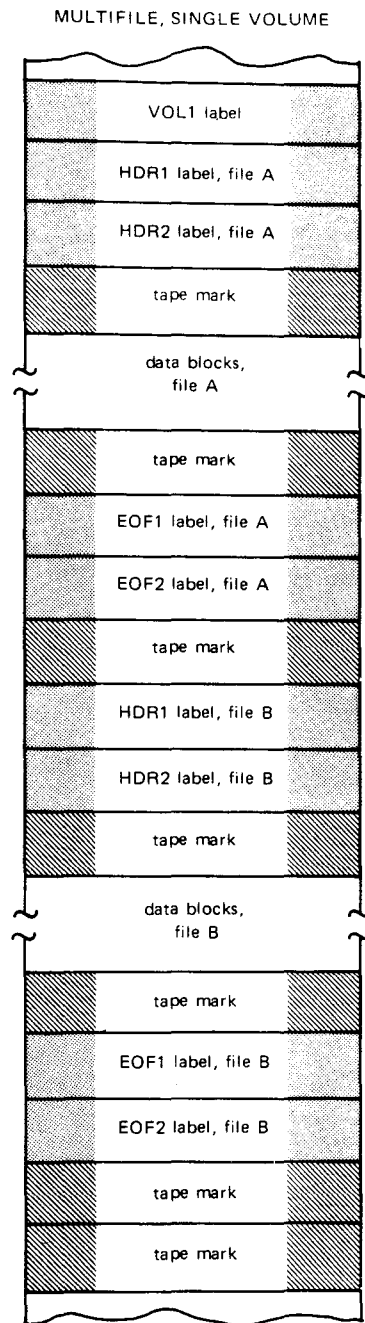


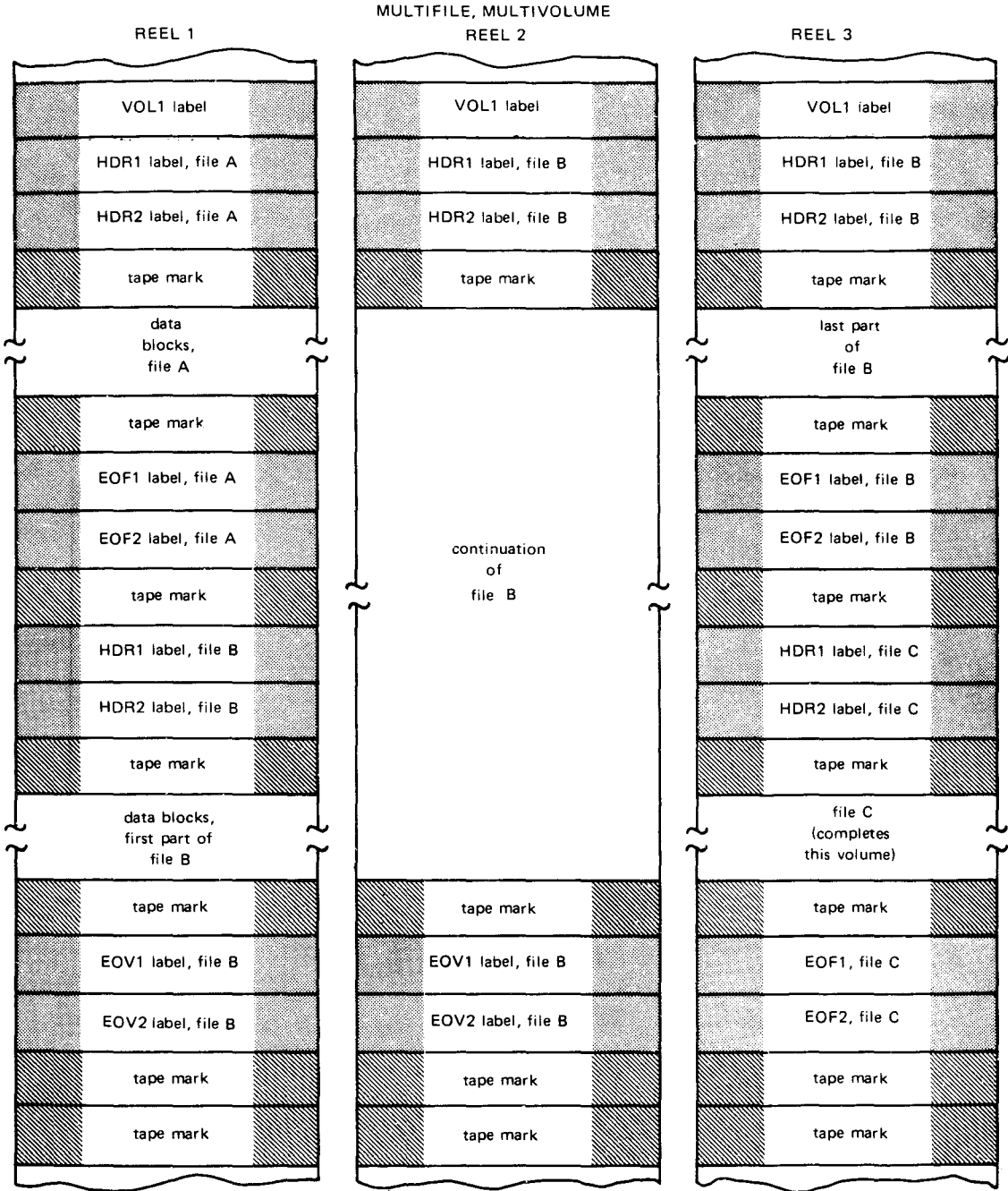
Figure 4-7. Organization of ASCII Single-File, Single-Volume and Single-File, Multivolume Sets



Legend

- Content supplied by user.
- These bytes are required and generated by data management.
- These bytes are generated by data management; user supplies data for certain fields.

Figure 4-8. Volume Organization of an ASCII Multifile, Single-Volume Set



Legend

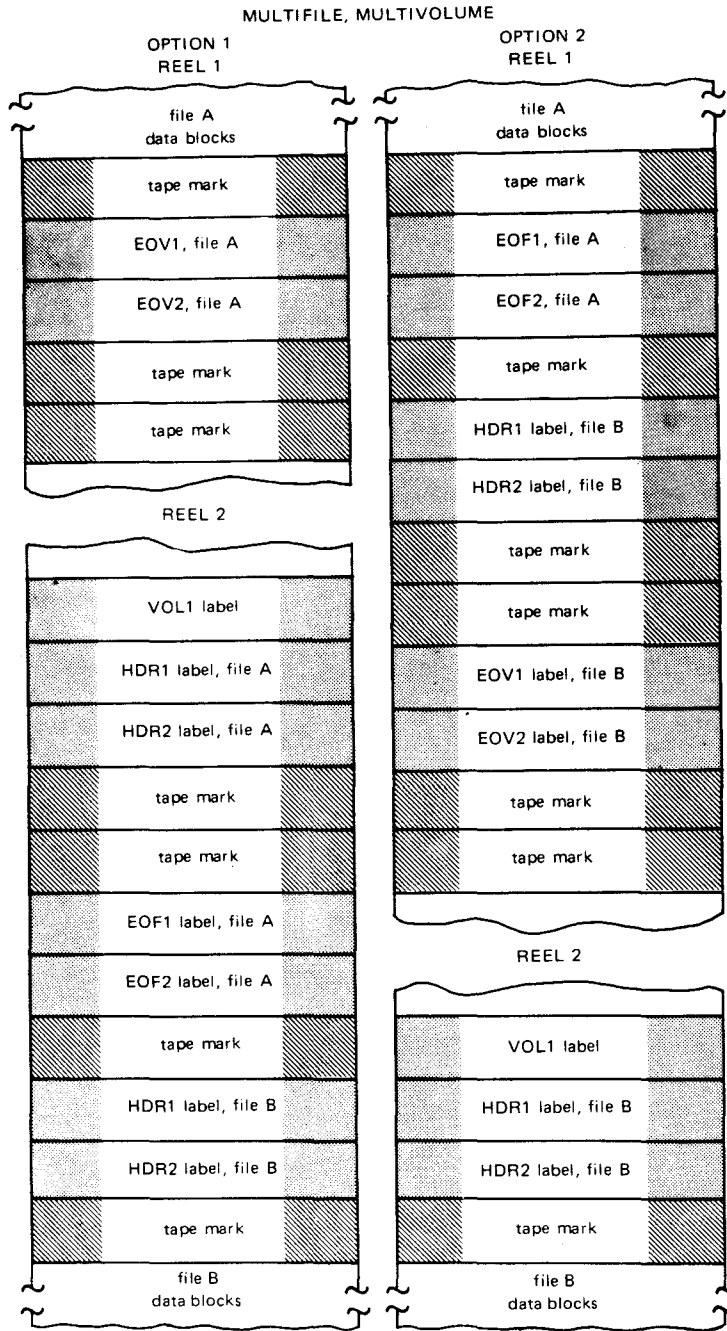
- Content supplied by user.
- These bytes are required and generated by data management.
- These bytes are generated by data management; user supplies data for certain fields.

Figure 4-9. Volume Organization of an ASCII Multifile, Multivolume Set

ASCII End-of-File and End-of-Volume Coincidence

The *American National Standard, X3.27-1978*, provides that whenever a volume ends within a file, the last block of the file is followed by an end-of-volume label (EOV1). It also allows for a second end-of-volume label (EOV2), which is standard in data management (an EOV1 label is always followed by an EOV2 label). A single tape mark precedes and two tape marks follow the EOV1 and EOV2 labels.

The standard also states that no file set may be terminated by end-of-volume labels; consequently, provision is made for those cases where the end-of-volume and end-of-file coincide. In these situations, the standard provides that the labeling configuration shall be one of the two options shown (see Figure 4-10). Option 1 occurs when the end-of-tape warning mark is reached while the last block of a file is being written. Option 2 occurs when the end-of-tape warning mark is reached after the EOF1 and EOF2 label groups have been started.



Legend

- Content supplied by user.
- These bytes are required and generated by data management.
- These bytes are generated by data management; user supplies data for certain fields.

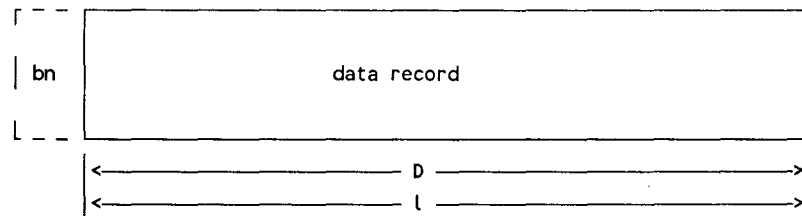
Figure 4-10. Label Configuration Options of an ASCII Multifile, Multivolume Set when End-of-Volume and End-of-File Coincide

4.2.5. Magnetic Tape File Record Formats

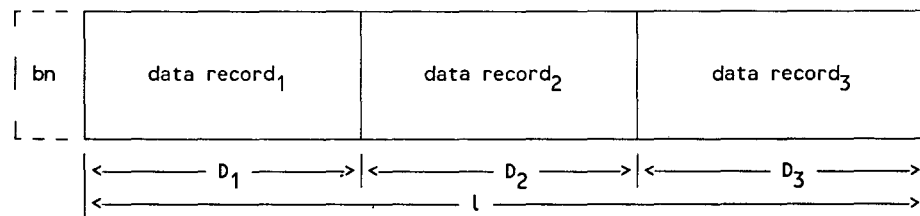
The data records on magnetic tape files may be fixed length, blocked or unblocked; variable length, blocked or unblocked; or undefined. Figure 4-11 shows these formats as they appear on EBCDIC and ASCII magnetic tape files. Note that the formats illustrated in Figure 4-11 do not show the optional use of padding because it is not supported. If your input blocks have been padded, the I/O area in your program must be large enough to accommodate this padding and your program should take care of detecting and removing the padding characters before it processes the data.

EBCDIC

FIXED-LENGTH UNBLOCKED RECORD



FIXED-LENGTH BLOCKED RECORDS



VARIABLE-LENGTH UNBLOCKED RECORD

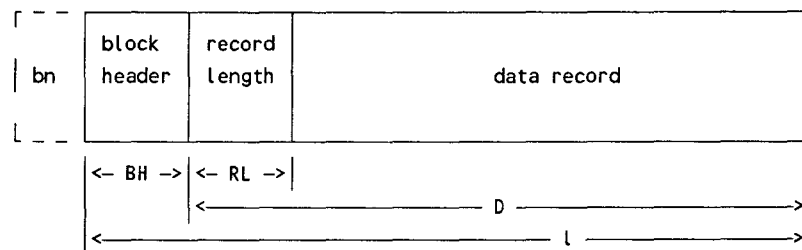
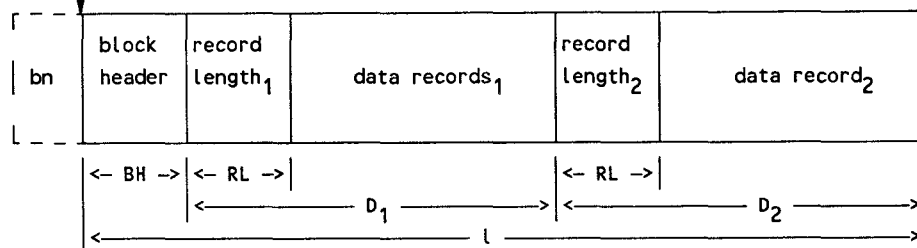


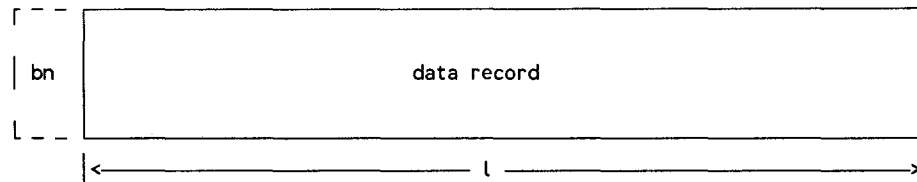
Figure 4-11. Record and Block Formats for Magnetic Tape Files, ASCII and EBCDIC (Part 1 of 4)

EBCDIC (cont.)

VARIABLE-LENGTH BLOCKED RECORDS

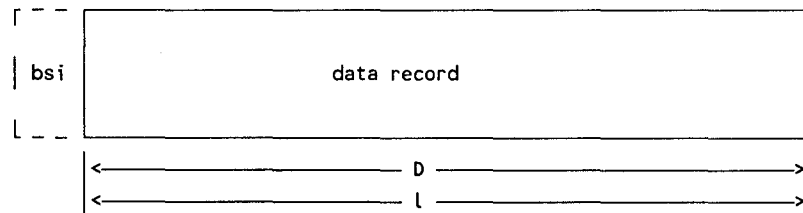


UNDEFINED RECORD FORMAT



ASCII

FIXED-LENGTH UNBLOCKED RECORD (FORMAT F)



FIXED-LENGTH BLOCKED RECORDS (FORMAT F)

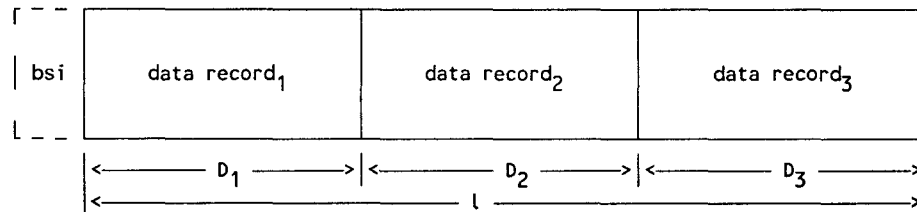
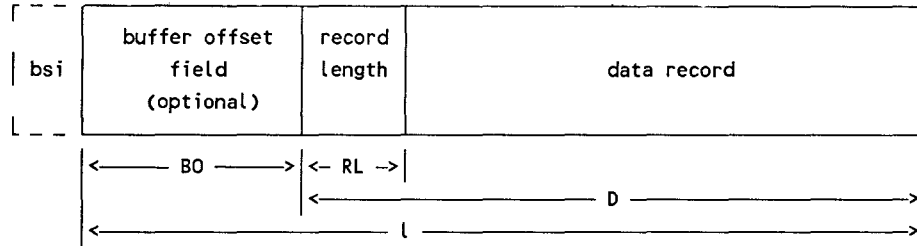


Figure 4-11. Record and Block Formats for Magnetic Tape Files, ASCII and EBCDIC (Part 2 of 4)

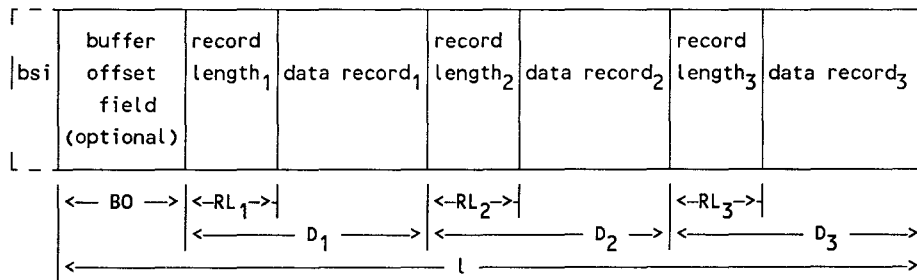
Magnetic Tape Formats and File Conventions

ASCII (cont.)

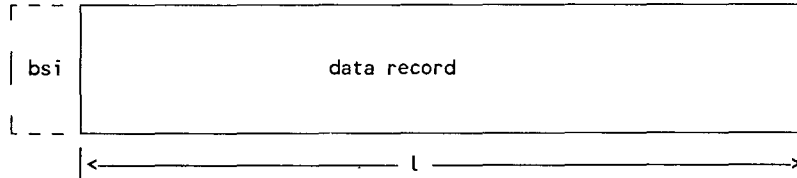
VARIABLE-LENGTH UNBLOCKED RECORD (FORMAT D)



VARIABLE-LENGTH BLOCKED RECORDS (FORMAT D)



UNDEFINED RECORD FORMAT (FORMAT U)



Legend

- D Record length, measured in bytes. This measure is entered in the most significant 2 bytes of the 4-byte record length field; the least significant 2 bytes are reserved.
- I Block length, measured in bytes. Minimum block length is 18 bytes. This measure is entered in the most significant 2 bytes of the 4-byte block header of EBCDIC variable-length records (blocked or unblocked); the least significant 2 bytes are reserved. When the buffer offset field of ASCII variable records is a 4-byte field, for output, data management writes the block length in it in ASCII. For input, data management assumes that it contains the length of the block in 4 bytes of ASCII.
- RL Record length field of variable-length records; a 4-byte field in ASCII and EBCDIC records. Its own length is included in the measure inserted here. In EBCDIC records, record length is read and written in binary; in ASCII records, it is recorded on tape in ASCII, although you present it to data management in binary and process it in binary.

Figure 4-11. Record and Block Formats for Magnetic Tape Files, ASCII and EBCDIC (Part 3 of 4)

- BH Block header, a 4-byte field at the head of the block format in which all EBCDIC variable-length records, blocked or unblocked, appear on magnetic tape. The most significant 2 bytes contain the length of the block, which includes the length of the header itself; the least significant 2 bytes are reserved.
- bn Optional 3-byte block number in EBCDIC. May not be created for output files. Data management accepts the block number in EBCDIC input files, but does not process it.
- BO Buffer offset field, an optional block prefix that may be placed at the head of each block of ASCII variable records. Its content is recorded in ASCII; its length ranges from 0 to 99 bytes for fixed and undefined records and is 0 or 4 for variable records. For variable records, when its length is 4 bytes, data management assumes that this field contains the length of the block (which includes the length of this field itself).
- bsi Optional 1-byte block sequence indicator for ASCII files in ASCII numeric code. May not be created for output files. Data management accepts the block sequence indicator in input files, but does not process it.
- ▼ The index register specified by the IOREG keyword parameter points here, to the first byte of the record length field of variable-length records.

Notes:

1. Although the *American National Standard, X3.27-1969*, also provides for a variable ASCII record with its record length specified in binary (commonly called the "V-format" record), data management does not support this format.
2. Spanned records (those extending beyond one block) are neither allowed in ASCII magnetic tape files (in which there must be an integral number of records per block) nor supported by data management in EBCDIC tape files.

Figure 4-11. Record and Block Formats for Magnetic Tape Files, ASCII and EBCDIC (Part 4 of 4)

4.3. Magnetic Tape File Job Control Considerations

The paragraphs that follow discuss the job control statements you include in your program execution job control stream when your program requires magnetic tape files. These are the DVC, VOL, LBL, and LFD statements (commonly called the device assignment set) and they must appear in this order in the job control stream.

The job control stream must contain a device assignment set for each magnetic tape file used by your program. This set is composed of at least one DVC (device assignment) statement and one LFD (logical file definition) statement. The set may also include a VOL (volume specification) statement and a LBL (file label information) statement when you need them to specify certain actions and information to data management.

See the *Job Control Language Programming Guide* (UP-9986) for the full formats of these statements and other details on their use.

4.3.1. Assigning a Tape Device to Your Job (DVC)

The DVC statement, with which you request the assignment of a tape unit to your job, must be the first in the device assignment set that you need for each magnetic tape file.

For the first positional parameter of the DVC statement, you specify *nnn*, a decimal number that is the logical unit number assigned to a tape device by your installation. The value of *nnn* normally ranges from 90 through 127 for tape units; however, your installation may have assigned other arbitrary logical unit numbers to one or more tape devices at system generation time. The two other specifications for the first positional parameter of the DVC statement (RES and RUN) are not used for magnetic tape files.

Your use of the optional second positional parameter of the DVC statement depends on your requirements; if you omit it, none of the three options available to a data management user apply. One option is to specify that job control may assign an alternative device of the same type as specified by *nnn*; for this, you specify ALT.

A second specification for this parameter, OPT, indicates that the device requested by this DVC statement is an optional device, not essential to the running of the job. If you select this option, you must indicate in your program that the associated file is an optional file.

The third specification for the second positional parameter of the DVC job control statement is IGNORE. You specify this form when you assign more than one logical file to the same tape device; that is, when your program accesses more than one of the files on a multivolume. You do not specify IGNORE when you are merely using the same tape unit for handling the different volumes of a multivolume file in succession.

The coding examples that follow illustrate some of the uses of the DVC job control statement in assigning devices to magnetic tape files. Note that the corresponding LFD statements, with which each DVC statement must be paired, are not shown. For a table of logical unit numbers and further details, refer to the *Job Control Language Programming Guide* (UP-9986).

Examples

	1	10	16	72
1.	// DVC 91			
2.	// DVC 122,ALT			
3.	// DVC 97,OPT			
4.	// DVC 111,IGNORE			

Explanations

1. Requests the assignment of any tape device to your job, for the file named in the accompanying LFD statement.

2. Requests the assignment of two tape devices of the same type for the file named in the accompanying LFD statement.
3. Requests the assignment of any tape device to this job. However, this device is not essential to the job. The job can be run even if this device is not available.
4. Requests the assignment of a tape device to this job. This same unit is also assigned to other files by LFD statements paired with identical DVC statements in the control stream for this job.

4.3.2. Specifying Tape Volume Information (VOL)

The VOL statement is required for all three types of files: standard labeled, nonstandard labeled, and unlabeled files. For standard labeled files, it is used to supply volume serial numbers (VSNs) for checking by data management.

For nonstandard and unlabeled files, you must include the (NOV) parameter. This is necessary because these files do not contain VOL1 labels and therefore do not have VSNs. When you use a VOL statement, you insert it in your device assignment set immediately after the DVC statement and before the LBL statement (if used).

Inhibiting Volume Serial Number Checking

Normally, for a standard labeled input file, you will want data management to check the VSN you specified in the first positional parameter of the VOL statement against the VSN contained in the VOL1 label in the tape volume. This action is performed when your file is opened. If you want to bypass this checking, you can do so by specifying (NOV) following the VSN as shown in the following example:

Example

```
// VOL ABC123(NOV)
```

Explanation

The example specifies that volume checking is not to be performed on the tape volume whose VSN is ABC123.

There are other ways in which volume checking can be specified or inhibited. These are described in detail in 4.3.3, which describes the LBL statement, and in Table 4-1.

4.3.3. Specifying Tape File Label Information (LBL)

You use the LBL statement as the next-to-last job control statement in the device assignment set for a standard labeled tape file to provide data management with the information it needs for writing and checking the file header label group (HDR1 and HDR2 labels) and the end-of-file label group (EOF1 and EOF2 labels) or end-of-volume label group (EOV1 and EOV2 labels). You must have it for volume checking; it has seven positional parameters, only the first of which is required.

Specifying the File Identifier

In the first positional parameter of the LBL statement, you must specify a file label that corresponds, not necessarily to the logical name of the file (by which your program refers to it), but to the file name that is recorded in the HDR1 label of the file on the volume. The file identifier for a tape file may contain as many as 17 characters, including blanks; if it contains blanks, you enclose it in apostrophes. If you do not have an LBL statement for an output file, data management uses the logical file name (specified in the mandatory LFD statement) for the file identifier field of the HDR1 and EOF1 or EOV1 labels.

Checking and Creating Volume and File Serial Numbers

In the second positional parameter of the LBL statement, you either specify the file serial number or request data management to check the relationship between this and the volume serial number on an input file or to create the file serial number on an output file. If you specify the file serial number, it is a 1- to 6-character alphanumeric string identical to the volume serial number of the first volume of the file. The option VCHECK in the second positional parameter requests data management to create or check this identity in the file labels. The VCHECK parameter instructs job control to use the volume serial number of the first volume of the file as the file serial number. If you have four volumes in a file, and you only want to use the last two volumes (3 and 4, in that order), you must specify the file serial number that is the same as the volume serial number of the actual first volume of the file. When the 4-volume file is created, you use the VCHECK parameter to assign the same file serial number to each volume. If you use the VCHECK parameter again, while trying to read only the third and fourth volumes, job control uses the volume serial number of the first volume you've coded (volume 3) as the file serial number value. Because volumes 3 and 4 were created with the file serial number of volume 1, the job will not run. Omitting the second positional parameter may inhibit the check or creation actions, depending on what you have specified in the VOL card and on whether the file is an input or an output file. Table 4-1 summarizes the combined effects of the various VOL and LBL statement specifications upon the actions taken by data management when a file is opened.

Table 4-1. Effects of Job Control VOL and LBL Statements on Data Management when a File Is Opened, Standard Labeled Tape File

VOL Statement, Specification of Positional Parameter 1	LBL Statement, Specification of Positional Parameter 2	Resulting Action by OPEN Transient		N o t e s
		VSN Field of VOL1 Label	File Serial Number Field of HDR1 Label	
Input Files				1
Unique VSN	Unique FSN	Checks that VSN in VOL1 label is identical to VSN specified in VOL statement	Checks that FSN in HDR1 label is identical to VSN in the VOL1 label of the first volume of the file	—
Unique VSN	VCHECK	Checks that VSN in VOL1 label is identical to VSN specified in VOL statement	Checks that FSN in HDR1 label is identical to VSN in the VOL1 label of the first volume of the file	—
Unique VSN	Blank, or statement omitted	Checks that VSN in VOL1 label is identical to VSN specified in VOL statement	No check made	—
Output Files				1
Unique VSN	Unique FSN	Creates VSN in VOL1 label identical to the VSN specified in the VOL statement, or checks that VSNs are identical	Creates FSN in HDR1 label identical to VSN in the VOL1 label of the first volume of the file	2
Unique VSN	VCHECK	Creates VSN in VOL1 label identical to the VSN specified in the VOL statement, or checks that VSNs are identical	Creates FSN in HDR1 label identical to VSN in the VOL1 label of the first volume of the file	2
Unique VSN	Blank, or statement omitted	Creates VSN in VOL1 label identical to the VSN specified in the VOL statement, or checks that the VSNs are identical	Creates FSN in HDR1 label identical to VSN in the VOL1 label of the first volume of the file	2

Notes:

1. Specifying any combination of the VOL and LBL statements other than those shown for input or output files is invalid and results in an error message being issued. Invalid job control specifications must be corrected and your job rerun.
2. When you specify (PREP) following the VSN in the VOL statement for an output file, you may specify any of these three combinations in the LBL statement. Refer to 4.3.5 for a description of the data management automatic tape prep facility.

Specifying the File Expiration Date

To provide data management with the expiration date of a file, you specify the expiration date in either of the two forms provided for it in the third positional parameter of the LBL statement. These forms are the actual expiration date you want or the number of days the file is to be retained beyond its creation date. This information is inserted in the file HDR1 label. If you use the file retention form of this option, you should either specify a file creation date or rely on the default assumption as described in "Specifying the File Creation Date" in this subsection. If you do not specify a file expiration date, data management makes the expiration date the same as the creation date.

Examples

```
1          10      16                               72
1. // LBL FICA,A12345,76366
2. // LBL FICA,B12345,R30
```

Explanations

1. Specifies that the expiration date of the tape file, whose external label is FICA and whose FSN is A12345, is 31 December 1976.
2. Specifies that the tape file, whose external label is FICA and whose FSN is B12345, is to be saved 30 days beyond the date it is created.

Specifying the File Creation Date

You may specify the creation date of the file by coding the fourth positional parameter of the LBL statement. If you omit this parameter, data management uses the date stored in the job prologue.

Specifying the File Sequence Number

The file sequence number is not the same thing as the file serial number; you use the file sequence number in a multifile tape volume to indicate the sequential position of a file with respect to the first file on the reel. The file sequence number of the first file may be 0001. Data management assumes this value is specified if you omit the fifth positional parameter in the LBL statement, and so records it in the file HDR1 label.

Specifying the File Generation and Version Numbers

You may specify a unique edition of a file by specifying its generation number in the sixth positional parameter of the LBL statement; the default assumption is 001. You may specify the version number of this generation in the seventh parameter; here, the default is 01.

4.3.4. Defining Your Logical File (LFD)

Each device assignment set in your job control stream must end with an LFD statement that specifies the logical file name for your file. This file name must be the same as the name you specified when you defined your file in your program. Job control allows the file name to be as many as eight alphanumeric characters. Data management, however, requires that the file name not exceed seven characters and the first character must be alphabetic.

The file name in the LFD statement is the first positional parameter. To ensure that a read-only type file is not inadvertently written over, you may precede the file name in the LFD statement with an asterisk (*) to indicate an input only file. This indicates that the operator should verify that the write-enable ring has been removed from the tape reel.

The second positional parameter of the LFD statement is not used in device assignment sets for magnetic tape files.

The EXTEND option of the third parameter may be used on the LFD statement to specify that a previously created output file is to be extended (see 4.3.8).

The coding examples that follow illustrate the uses of the LFD statement.

Examples

1	10	16	72
<hr/>			
1.	//	LFD *YOURFLE	
2.	//	LFD MYFILE	
3.	//	LFD OUTFLE,,EXTEND	

Explanations

1. Specifies that the logical name of the file to which this device assignment set pertains is YOURFLE, and that it is an input-only file. The operator should verify that the write-enable ring has been removed from the tape reel.
2. Specifies that the logical name of the file to which this device assignment set applies is MYFILE. It is not designated as an input-only file, and no physical safeguard is made by the operator against its being overwritten.
3. Specifies that the output tape file, OUTFLE, is to be extended. OUTFLE must be the last or only file on a single tape volume or a file that needs additional space on a new volume.

4.3.5. Preparing Tape Volumes

If you intend to create standard labeled tape files in your program, you must first prepare the tape volumes on which you intend to write your output files. This process consists of writing the standard VOL1, HDR1, and HDR2 labels on these tape volumes. There are two ways you can do this. The first way is to use the tape prep utility (TPREP) to prepare the volumes before they are used in your program. This is described in the *System Service Programs (SSP) Operating Guide* (UP-8841).

The second way is to request data management to prepare the required tape volumes for the file at program execution time. This is accomplished by specifying (PREP) in the VOL statement for your file after the VSN of each volume that is to be prepped. When you do this, data management writes the standard VOL1, HDR1, and HDR2 labels on these volumes when the file is opened and processing continues from that point on.

Part of the information required by this tape prep facility for writing HDR1 and HDR2 labels may be specified in the LBL statement; therefore, when you specify (PREP) on the VOL statement of a device assignment set, you may also include an LBL statement in the set (refer to 4.3.3 and Table 4-1).

The examples that follow show how to use the VOL statement to cause data management to prep tape volumes at program execution time.

Examples

1	10	16	72
---	----	----	----

```
1. // VOL ABC789,DEFO12(PREP)
2. // VOL ,,,,DEFO14(PREP)
```

Explanations

1. Calls on data management to prep two tape volumes, whose VSNs are ABC789 and DEFO12. By default, data management uses the tape density and other mode characteristics specified by your installation at system generation for the tape device specified in the DVC statement which starts this device assignment set. Data management uses information supplied in the accompanying LBL statement to write the HDR1 and HDR2 labels for the files on these volumes.
2. Calls on data management to prep a tape volume (DEFO14), but set the volume sequence number (in the HDR1 label) to 4 rather than 1. The prep utility adds one for every comma immediately preceding the first operand to get the volume sequence number. Whenever you use tapes in a multivolume file environment, this procedure should be used because it eliminates the use of scratch volumes. If scratch volumes are used, the volume sequence number defaults to 1, resulting in a volume sequence number error condition. Even though this condition can be ignored, it is recommended that your volume sequence numbers be valid.

4.3.6. Specifying Mode Characteristics for Tape Volumes

Data management does not set the mode characteristics of the tape volumes, such as bytes per inch (recording density), parity, and the number of tracks. These characteristics are usually set at system generation and all tape volumes are recorded with these characteristics by data management.

If you need to use or produce a volume that does not conform to the mode characteristics set at system generation, you must specify this via the Mcc parameter in the VOL statement for this volume in the device assignment set. The following example shows how this can be done:

Example

```

1      10      16      72
-----
// VOL  MC0,ABC123

```

Explanation

This example specifies that the tape volume whose VSN is ABC123 has a recording density of 1600 bytes per inch and parity is odd. For full details on mode characteristics, refer to the *Job Control Language Programming Guide* (UP-9986).

4.3.7. Creating Multivolume Files

As explained in 4.3.5, you may use the VOL job control statement to specify the volume serial numbers (one for each tape used) of the volumes that will constitute a magnetic tape output file. If you have not previously prepped these tapes, specification of the (PREP) parameter in the VOL statement causes them to be prepped automatically for you as part of the job step creating the file. These procedures are helpful when you know precisely the size of the file and the exact number of tapes it requires - but this is not always the case with a new file.

If you specify (PREP) and fill all the tapes specified in the VOL statement and have not completed creating your file, there is no way to complete it in this job step. An error message is issued (DM41 FILE SPACE IS EXHAUSTED) and your job cancels. In some circumstances, you do not know exactly which was the last block written to this tape. Therefore, you may still not be able to estimate easily how many volumes are needed in your next effort to create the file. In other circumstances, possibly the entire run has been futile.

Data management offers a useful facility in case you do not know the exact number of volumes needed for an output file. It automatically extends your file, one volume at a time, after the last volume specified on the VOL statement has been exhausted. It continues to do this until you terminate your file creation operation. This can only be done, however, if you do not specify the (PREP) parameter in the VOL statement. Automatic prepping of tapes cannot be requested at the same time the automatic extension feature is requested. Furthermore, if you are creating a nonstandard tape file, the (NOV) parameter must be specified in the VOL statement.

When you are creating a file under these conditions, data management issues a mount message to the operator for a scratch volume each time an end-of-volume marker is reached. This continues until you terminate the file creation operation. The message issued to the operator at the system console is

```
MOUNT VSN=SCRATCH ON DVC xxx RIC
```

When this message is displayed, the operator must mount either a preprepped tape, if a standard labeled file is being created, or an unprepped tape, for a nonstandard labeled or unlabeled file, and then reply R to the mount message to allow the program to continue writing your output file but on the new volume. If a standard labeled file is being created, you should make enough preprepped and properly sequenced tapes available to allow the operator to complete the operation as requested. You must also furnish the operator with instructions designating the order in which the tapes are to be mounted. If the tapes are mounted out of sequence and are correctly preprepped with consecutive volume sequence numbers, the output file is created, but an error message indicating the out-of-sequence condition is displayed on the system console each time the file is read. Although these messages may not confuse you, they could cause other users of the file unnecessary concern. If a nonstandard labeled or unlabeled file is being created, your instructions should request the operator to mount an unprepped volume each time the mount message is displayed and somehow identify the order in which they are to be mounted.

4.3.8. Extending Tape Files

Data management allows you to extend a standard labeled tape file provided that the file is not physically followed by another file on the same volume. When you extend a file, the record size, record format, and block size specifications of your extending program must be the same as those specified for the original file. Your extending program must also specify that the file is an output-only file.

Both single-volume and multivolume files can be extended by specifying the EXTEND parameter in the LFD statement associated with the file. The EXTEND parameter directs data management to proceed to the end of the file when the file is opened and erase the end-of-file label. The extension then begins on the mounted tape unless the end-of-file label happens to coincide with the end-of-volume marker. If this happens, a mount message is issued. In either case, data management proceeds to extend your

file until it is closed by your extending program. If an end-of-volume condition is reached and there is more data to be placed on the file, data management requests the mounting of the next volume specified in the VOL statement for this file and continues extending it.

If you think that your file extension operation will require the operator to mount additional tapes, you should prep the volumes you want used with the appropriate volume sequence numbers before you begin your file extension operation. This allows you to identify the tapes you want to use in the VOL statement, and eliminates the confusion that arises when the tape file is read and the volume sequence numbers on the tapes are not consecutive. Remember that you cannot request automatic prepping of tape volumes while you are extending a tape file. This can only be requested with a file creation operation. Consequently, the (PREP) cannot be specified in a VOL statement associated with a file extension operation.

The following device assignment set shows how you specify the extension of a file that is not expected to exceed the limits of the present tape volume:

```
// DVC 90
// VOL TP01
// LBL MASTER
// LFD MAST,,EXTEND
```

If you expect the extension to exceed the space remaining on the present volume, you should add the VSNs of the new tapes to the VOL statement as follows:

```
// DVC 90
// VOL TP01,TP02,TP03,...
// LBL MASTER
// LFD MAST,,EXTEND
```

If you are extending a multivolume tape file, your device assignment set need only identify the last volume containing the file (plus any new volumes you may need), but you must also include the VSN of the first tape volume in the file as the second parameter in the LBL statement. For example, if you had a master file on tapes TP01, TP02, and TP03 that you wanted to extend, and you expected your extension to require one additional tape, your device assignment set would be as follows:

```
// DVC 90
// VOL ,,,TP03,TP04
// LBL MASTER,TP01
// LFD MAST,,EXTEND
```

The first VSN (TP01) is required to identify the new tape TP04, as well as TP03, as being part of a file named MAST, which begins on TP01.



Section 5

Disk and Format Label Diskette Formats and File Conventions

5.1. General Information

Whenever "disk file" is used in the following discussion, it refers to both disk and format label diskette files. A format label diskette file is treated exactly the same as any other disk file.

Disk files consist of data records that are recorded on one or more volumes (disk packs). These files differ from other types of files in that the data records can be retrieved not only sequentially but also randomly by relative record number (the position of a record in the file relative to the beginning of the file) or by the record key (a character string specified within each record to uniquely identify that record). The data records are retrieved or written on the disk volume via a disk subsystem. Refer to Appendix A for the functional characteristics of the disk subsystems that are supported.

Usually, there is more than one file on a disk volume. In order to keep track of where these files are located, each volume contains a volume table of contents (VTOC). Each file also contains system standard labels that identify and delineate it.

5.1.1. How Disk Files Are Organized

Disk files fall into two general categories: nonindexed and indexed.

A nonindexed file is organized consecutively. Its records are written on the disk in the order in which they are presented. The records are processed consecutively in the same order as they appear on the disk. A nonindexed file can also be one that is organized relatively; each record in the file is written on the disk in a specific position relative to the beginning of the file. This allows any record in the file to be retrieved directly without processing any preceding records when the location (relative record number) of the record is specified.

An indexed file contains data records and an index of the record keys. The data records appear on the disk in the order in which you submitted them and the index is arranged in ascending key order. The records can be processed sequentially or randomly by record key. If the records are processed sequentially, the processing commences with the record that has the lowest key value. For random retrieval, you need only specify the key of the record you want retrieved.

5.1.2. Disk Access Method

Consolidated data management uses one access method for all disk files - MIRAM (multiple indexed random access method). MIRAM provides the functions that were previously provided by separate access methods.

MIRAM Concepts

MIRAM has a number of features and concepts that distinguish it from other disk access methods.

- Each data record is stored in a record slot. Record slots in MIRAM files, for either fixed- or variable-length records, are of uniform size and may span physical blocks, tracks, and cylinders as required. They may even extend from one volume to another (unless the file was created for processing only a single volume online at a time).
- Data record slots are written on disk compactly as a continuous string of bytes.
- The data buffer size specified in the program only determines the number of physical blocks that are transferred between the disk and main storage in a single access.
- The string of data records can always be accessed sequentially (consecutively) or by relative record number. In addition, the data can be indexed by up to five keys; this causes MIRAM to build a suitable index structure for each key type in a partition that is separate from the data.
- An indexed MIRAM file can be accessed by the additional random-by-key or sequential-by-key modes using a given key of reference, which can be changed.
- Indexed MIRAM files, either multivolume or single volume, may be created by means of an orderly load (records submitted in ascending key order) or a disorderly load (records submitted in no particular key order) and they may be extended by appending records in either manner. MIRAM does not sort the index at the completion of a disorderly load, but maintains the index current on a record-by-record basis.
- Duplicate keys are permitted.
- A new record is immediately available for retrieval whether it has been added to an indexed or nonindexed file.
- Multivolume MIRAM files may be created for processing with either one volume online at a time or with all volumes online. They must be processed in the same manner as they were created. All volumes must be the same device type.

- All programs that access a MIRAM file need not use the same data buffer size for input/output as was used to create the file. Those that access an indexed MIRAM file, however, must use the same index buffer size.
- MIRAM allows you to logically delete records in your files; that is, it allows you to mark records so that in subsequent processing they will be ignored. A deleted record count is maintained for all files created under Release 12.0 or later. This count will always be accurate for files created with recovery. However, for files without recovery, the count will show the number of deleted records at the time the file was last closed.

The deleted record count is displayed on the System Utilities (SU) 'VTP' print. Files that are created with this functionality will show a valid count; that is, zero, or the actual count. When an accurate count is not available (for example, a file created prior to Release 12.0), the field will be blank.

- MIRAM always extends an existing file unless the INIT parameter is specified on the LFD statement. This causes initialization of the file. MIRAM does not recognize the EXTEND parameter on the LFD statement, and you cannot use EXTEND to override a program specification that calls for initialization of the file.
- These are the restrictions.
 - The maximum key length is 80 bytes.
 - No byte of a record key may contain the hexadecimal value 'FF'.
 - The minimum size for the index and data buffer is 256 bytes.
 - The maximum size for the index buffer is 32,512 bytes.
 - You may not mix device types within a multivolume file set.
- The two types of MIRAM files are MIRAM characteristic files and IRAM characteristic files. MIRAM characteristic files meet one or more of the following conditions:
 - Multiple keys
 - Duplicate keys permitted
 - Key change on update permitted
 - Variable file record format
 - Record control byte (RCB)

You may use the `MIRAMCHAR SYSGEN` parameter to specify `MIRAM` characteristics for all newly created `MIRAM` files, regardless of the file's actual characteristics. For further details on the `MIRAMCHAR` parameter, refer to the *Installation Guide* (UP-8839).

5.2. `MIRAM` File Organization

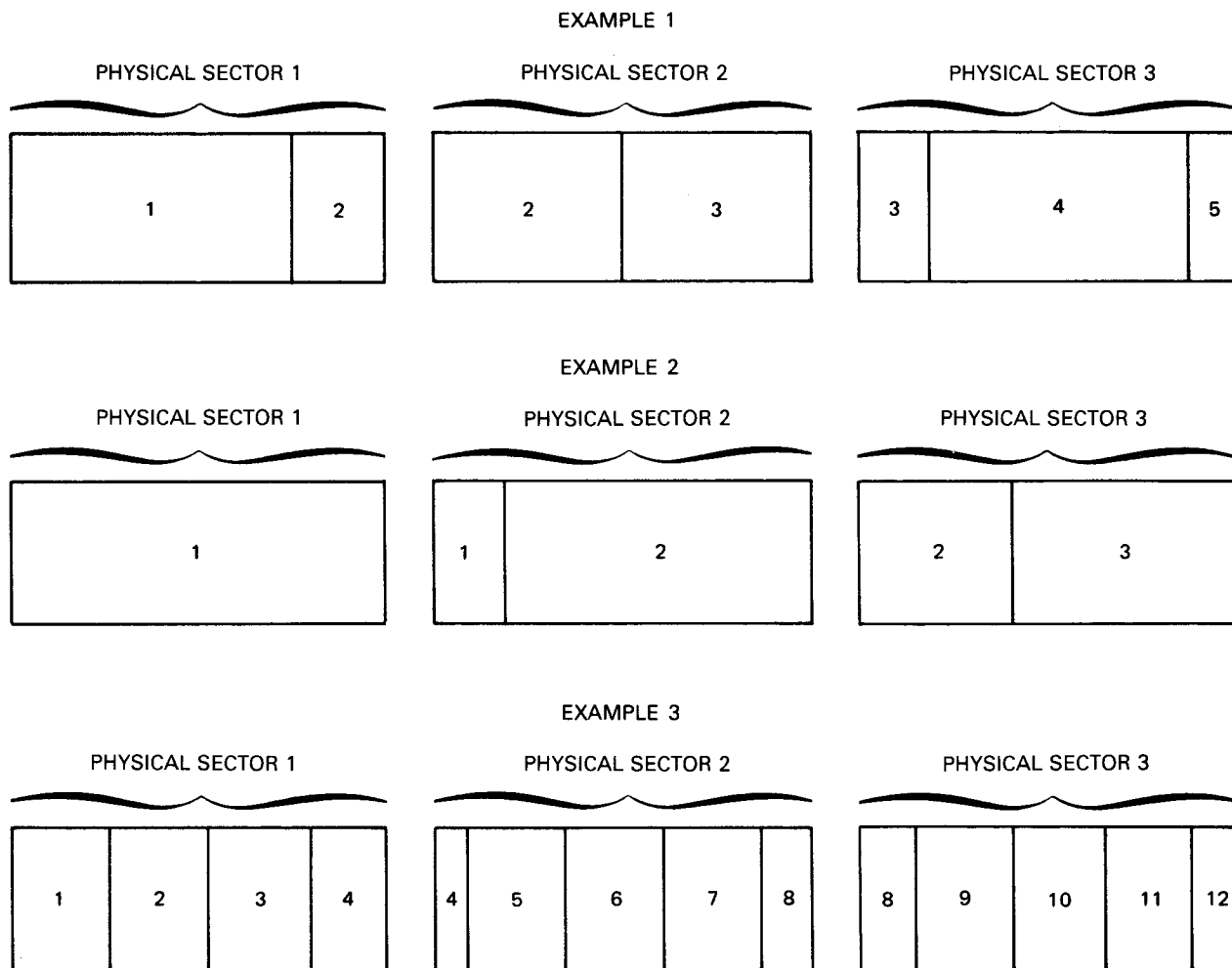
All `MIRAM` files contain two partitions: the data partition, which contains the data records, and the index partition, which contains an index for each of the keys in your records. If the file is a nonindexed file, the index partition is not used; that is, no entries are placed in it and no space is allocated to it. If the file is indexed, entries are placed in the index partition and space is allocated to it.

For indexed files, the data partition precedes the index partition.

5.2.1. The Data Partition

The data partition is arranged in the same way for both nonindexed and indexed files. It consists of a single compact string of data records that may be keyed or unkeyed.

When data records are stored in a `MIRAM` file, the records are placed in uniform size record slots and are arranged in the same order in which you originally presented them. These data records are stored in 256-byte physical sectors on your disk packs. Because the record slot size does not have to conform to the physical sector size, the records may span these physical boundaries, as shown in Figure 5-1.



Notes:

1. All physical sectors are 256 bytes.
2. 1, 2, 3, ... n represent record slots.
3. Record slots in Example 1 are approximately 190 bytes each.
4. Record slots in Example 2 are approximately 300 bytes each.
5. Record slots in Example 3 are approximately 70 bytes each.

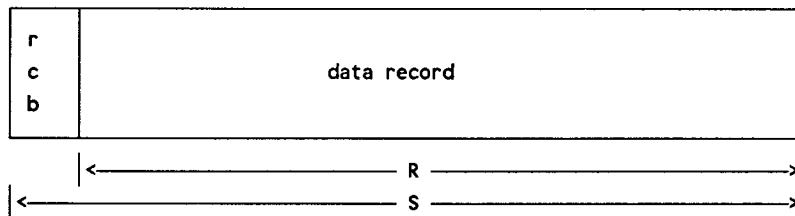
Figure 5-1. Disk (MIRAM) Data Record Slots Spanning Physical Sector Boundaries

Disk and Format Label Diskette Formats and File Conventions

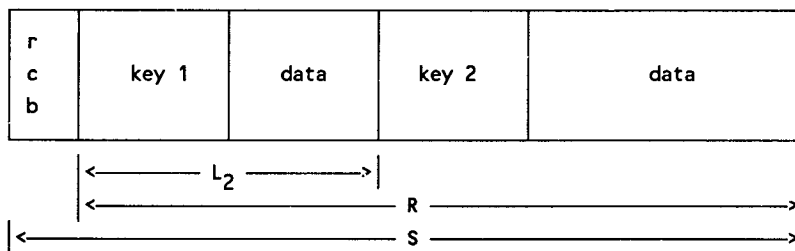
Your data records may also span track boundaries, cylinder boundaries, and volume boundaries (except when a multivolume file is created for processing with only one volume online at any one time). When new records are added to a file, they are appended to the existing data record string; that is, they are added at the end as a continuation of the original string.

The formats of the disk data records are shown in Figure 5-2.

FIXED-LENGTH RECORDS WITHOUT KEYS



FIXED-LENGTH RECORDS WITH KEYS



VARIABLE-LENGTH RECORDS WITHOUT KEYS

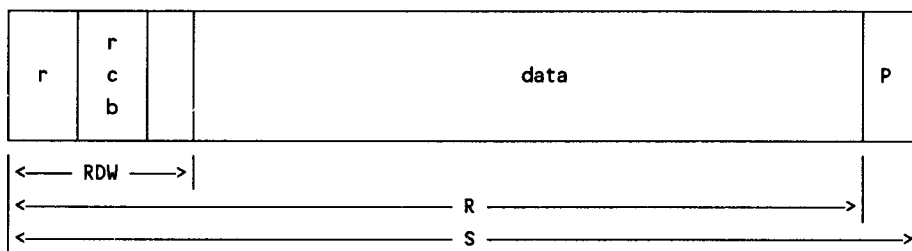
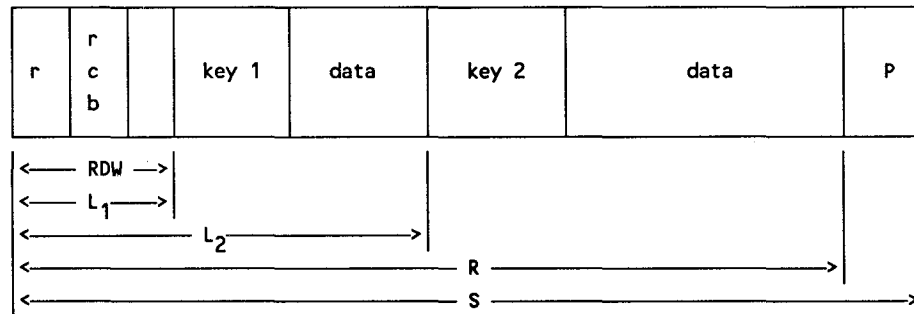


Figure 5-2. Disk (MIRAM) Data Record Formats (Part 1 of 2)

VARIABLE-LENGTH RECORDS WITH KEYS



Legend

- rcb** Record control byte (optional). Used to indicate that a record has been logically deleted from the file. For MIRAM fixed-length records, this byte is placed at the beginning of each record. For variable-length records, the third byte of the record descriptor word (RDW) is used as the rcb.

- R** Length of the logical record (RDW plus keys plus data). You specify this length as the number of bytes. For variable-length records, this value, expressed in binary, must be placed in the first 2 bytes of the RDW.

- RDW** 4-byte record descriptor word for variable-length records. The first 2 bytes contain the logical record length (r) expressed in binary; the third byte may be used as the rcb; the fourth byte is not used.

- L_n** The starting location of record key n (n = 1 through 5) of a MIRAM file data record when the key does not start in the first byte of the record. L_n represents the number of bytes (RDW plus data) that precede key n. The starting location of key n must be the same in each record. Key n must have the same length in each record (a minimum of 1 byte and a maximum of 80), and no byte may contain the hexadecimal value FF.

- S** Slot size. All records are written into fixed-size slots. Slot size equals the record size + 1 for fixed-length records with an rcb; otherwise, slot size equals the record size. For variable-length records, the slot size equals the maximum record size.

- P** Padding.

Figure 5-2. Disk (MIRAM) Data Record Formats (Part 2 of 2)

5.2.2. Entries in the Index Partition

If you have keyed records, entries are placed in the index partition as these records are loaded into the data partition. MIRAM extracts all the keys from each record (a maximum of five keys is permitted) and constructs a 3-byte pointer for each of the keys from the file-relative record number of the position the record was written to. From these, it forms an index entry for each of the keys in the record and stores them in the index partition. The index entry for each key consists of the key plus 3 bytes (equal to the specified key length plus 3 bytes) and is stored in an area of the index partition, which is called a fine-level index. If you have three keys in each record, the index entry for each key is stored in a separate fine-level index; that is, the entry for key 1 is stored in the fine-level index for key 1, the entry for key 2 is stored in the fine-level index for key 2, the entry for key 3 is stored in the fine-level index for key 3, and so on.

A fine-level index is not formatted for hardware search, unlike the other levels of index that are described subsequently. It is treated as a chain of multisector blocks where each sector is 256 bytes long. All entries in a fine-level index are maintained in ascending key order. Figure 5-3 shows a typical fine-level index block of three sectors.

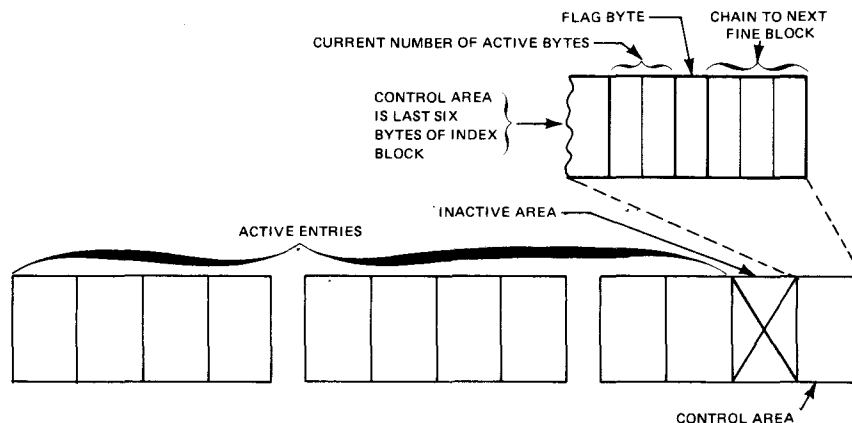


Figure 5-3. Fine-Level Index Block

When a fine-level index is created, another hierarchical level of index is always created - the coarse-level index. This is hardware searchable and is composed of 256-byte blocks that contain entries similar to those in the fine-level index. They differ, however, in that the 3-byte pointer in each coarse-level entry does not represent the file-relative number of a record in the data partition. Instead, it points to another index block at a lower level - either a fine-level block or a block in what is called a mid-level index.

Another difference is that instead of having a 6-byte control area, each coarse-level block uses its final byte to indicate the number of active entries. The high key of the block is the first one encountered by the hardware search. Both the coarse-level and mid-level index blocks have the same format (see Figure 5-4):

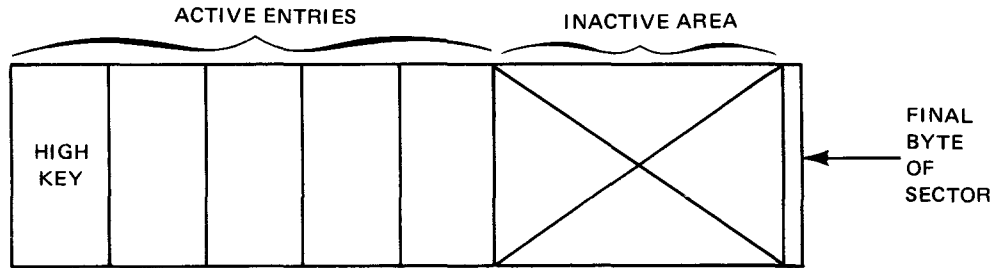


Figure 5-4. Coarse-Level or Mid-Level Index Block

5.2.3. MIRAM Index Structure

As you know, you can specify up to a maximum of five keys for a file. For each key that you specify, MIRAM builds a separate index structure. In those files where you have more than one key, these separate index structures allow you to use any of the key types as the key of reference to access your data records when you subsequently use the file in a program.

When MIRAM builds an index structure for your file, it creates a minimum of two levels of index: a fine-level index and a coarse-level index. If your file is very large, one or more mid-level indexes are created as needed. The fine-level index consists of one entry for every record in the data partition of your file. The fine-level entries are filed in ascending key order until an index block (256 bytes) is filled. At this time, one coarse-level entry is made that contains the high key entry of that filled block of the fine-level index. As each fine-level index block is filled, another coarse-level entry is made. This process continues until all your records are on file.

The coarse-level index is automatically allocated by MIRAM. If the coarse-level index is filled before all your records are on file, a mid-level index is created.

5.2.4. Estimating Disk Space Required for an Indexed Disk File

You can use either cylinder or track allocation to allocate space for an indexed file.

The following procedure allows you to estimate the number of cylinders or tracks for your primary allocation of disk space to an indexed file. The result is an approximation you can use in specifying the EXT statement in the job control device assignment set that allocates disk space for an indexed file to be created by your program. This procedure can also be used to determine the number of cylinders or tracks to be allocated for an indexed file that is to be generated from another file by the data utility program.

The number of cylinders or tracks required for an indexed file includes those occupied by the data partition and the index structures for each key type in the file. To estimate the number of cylinders or tracks required by the file, proceed as follows:

First, calculate D, the number of sectors required for your data records; that is, the data partition.

Step 1

$$D = \frac{\text{record-length} \cdot \text{number-of-records}}{S}$$

where:

S is the physical sector size. The default size is 256 for all types of disk subsystems. For the 8430 and 8433 disk subsystems, this value can be greater than 256. (See VSEC parameter in Appendix C.)

Next, calculate B_i , the number of index blocks required by your fine-level index for key_i .

Step 2

$$B_i = \frac{\text{number-of-records} \cdot (\text{keylength}_i + 3) \cdot 4}{(256 \cdot m) \cdot 6} \cdot 3$$

where:

The factor of 4/3 is used because the average fine-level index will be 3/4 full.

m is the number of 256-byte sectors in the index buffer.

Then calculate F_i , the number of 256-byte sectors required by your fine-level index for key_i .

Step 3

$$F_i = m \cdot B_i$$

Repeat steps 2 and 3 as many times as necessary and then calculate F, the number of 256-byte sectors required by your fine-level indexes for all keys in the file.

Step 3a

$$F = \sum_{i=1}^n F_i$$

where:

n is the number of keys in the file.

Then perform the final calculation. This calculation, which is the sum of the data requirements and the fine-level index requirements, represents over 95 percent of the space required for an indexed file. Once this is determined, it is a simple matter to figure out what your space requirements are for a given file.

As you can see, formulas are provided for cylinder allocation and for track allocation. You can use any formula, but track allocation should only be used for files whose space requirements are ≤ 2 cylinders.

Step 4

Cylinder Allocation

$$C = \frac{F}{U \cdot N} + \frac{D}{A \cdot N}$$

or

$$C = \frac{T}{N}$$

Track Allocation

$$T = \frac{F}{U} + \frac{D}{A}$$

or

$$T = C \cdot N$$

where:

C
Is the number of cylinders to allocate to the file.

- A** Is the disk dependent number of sectors per track for the data partition. If the physical sector size is 256 (default), use Table 5-1 for this value. If the VSEC parameter (see Appendix C) is used to specify a larger physical sector size for an 8430 or 8433 disk subsystem, the number of sectors per track will have to be determined.
- D** Is the number of 256-byte sectors required for the data partition.
- F** Is the number of 256-byte sectors required by all the fine-level indexes in the file.
- U** Is the disk-dependent number of 256-byte sectors per track for the index partition (see Table 5-1).
- N** Is the disk-dependent number of tracks per cylinder (see Table 5-1).
- T** Is the number of tracks to allocate to the file.

Example

Assume that you want to calculate the number of cylinders to allocate for an indexed disk file and the following conditions apply:

Number of records	77,500
Record length	512 bytes
Keylength ₁	28 bytes
Keylength ₂	30 bytes
Sector size (data partition)	256 bytes
Index buffer length	512 bytes
Type of disk	8433

$$\begin{aligned}
 \bullet \quad D &= \frac{\text{record length} \cdot \text{number-of-records}}{256} \\
 &= \frac{512 \cdot 77,500}{256}
 \end{aligned}$$

= 155,000 sectors for data partition.

- $$B_1 = \frac{\text{number-of-records} \cdot (\text{keylength}_1 + 3) \cdot 4}{(256 \cdot m) - 6} \quad \frac{4}{3}$$

$$= \frac{77,500 \cdot (28 + 3) \cdot 4}{(256 \cdot 2) - 6} \quad \frac{4}{3}$$

= 6,331 index blocks required for the fine-level index for key₁.

- $F_1 = m \cdot B_1$

$$= 2 \cdot 6331$$

= 12,662 sectors for the fine-level index for key₁.

- $$B_2 = \frac{\text{number-of-records} \cdot (\text{keylength}_2 + 3) \cdot 4}{(256 \cdot m) - 6} \quad \frac{4}{3}$$

$$= \frac{77,500 \cdot (30 + 3) \cdot 4}{(256 \cdot 2) - 6} \quad \frac{4}{3}$$

= 6,739 index blocks required for the fine-level index for key₂.

- $F_2 = m \cdot B_2$

$$= 2 \cdot 6739$$

= 13,478 sectors for the fine-level index for key₂.

- $$F = \sum_{i=1}^n F_i$$

$$= F_1 + F_2$$

$$= 12,662 + 13,478$$

= 26,140 sectors for all the fine-level indexes for all keys in the file.

Disk and Format Label Diskette Formats and File Conventions

$$\begin{aligned}
 \bullet \quad C &= \frac{F}{U \cdot N} + \frac{D}{A \cdot N} \\
 &= \frac{26,140}{29 \cdot 19} + \frac{155,000}{33 \cdot 19}
 \end{aligned}$$

= 295 cylinders are required for the data and fine-level indexes.

Note: *After you have calculated your disk space requirements and you proceed to create your file, you must provide enough volumes to hold the file. This must be considered because the amount of space available is not the same for all disk types. Refer to Table A-4 to determine how many volumes you will need based upon your calculations and the type of disk you intend to use.*

The minimum cylinder and track allocation for indexed MIRAM files is described in 5.2.6.

Table 5-1. Disk-Dependent Factors for Estimating Disk Space Requirements

Unisys Disk Subsystem	U (Number of 256-byte sectors per disk track for index partition)	A (Number of 256-byte sectors per disk track for data partition)	N (Number of tracks per disk cylinder)
8416*	40	40	7
8418*	40	40	7
8417	60	60	14
8419	50	50	7
8430*	29	33	19
8433*	29	33	19
8470	96	96	32
8494	96	96	20

*Models 8-20 only

5.2.5. Estimating Disk Space Requirements for a Nonindexed MIRAM File

The following procedure allows you to estimate the number of cylinders or tracks required for your primary allocation of disk space to a nonindexed file.

As you can see, formulas are provided for cylinder allocation and for track allocation. You can use any formula, but track allocation should only be used for files whose space requirements are ≤ 2 cylinders.

<p><u>Cylinder Allocation</u></p> $C = \frac{D}{A \cdot N}$ <p>or</p> $C = \frac{T}{N}$	<p><u>Track Allocation</u></p> $T = \frac{D}{A}$ <p>or</p> $T = C \cdot N$
---	--

The calculation of D, A, and N is described in 5.2.4.

The minimum cylinder and track allocation for nonindexed MIRAM files is described in 5.2.6.

5.2.6. Minimum Cylinder and Track Allocation for MIRAM Files

Table 5-2 shows the minimum number of cylinders or tracks that are required by nonindexed and indexed MIRAM files.

Table 5-2. Minimum Cylinder and Track Allocation for MIRAM Files

File Type	Allocation Type	
	Cylinders	Tracks
Nonindexed	1	1
Indexed	2	6 + n*

*n is the number of keys in the file.

5.3. Disk File Job Control Considerations

The device assignment set that you use for a disk file in your program execution job control stream depends on how the file is being used in your program. If you are creating a disk file, you need a specific device assignment set for that situation, and if you are using an existing disk file, a different device assignment set is needed. In addition, you must also prepare a disk volume before you use it. These situations are discussed in the paragraphs that follow.

5.3.1. Device Assignment Set for Creating a Disk File

The device assignment set for creating a disk file consists of the DVC, VOL, EXT, LBL, and LFD job control statements in that order. The DVC statement supplies the device number, the VOL statement supplies the volume serial number (VSN), the LBL statement supplies the file identifier, and the LFD statement associates the file name you used in your program with the disk volume.

As you have noticed, an EXT statement is included in the device assignment set. This statement is required when you are creating a disk file. It is required because you have to allocate space for your file on the disk volume. The EXT statement describes your space requirements on the disk. The question of how much space is required for a file can be determined by using the formulas that are shown in 5.2.4 and 5.2.5. When you use the EXT statement, the first positional parameter specifies the access method. Because we are dealing with a MIRAM file, this parameter is always MI. Because this is the default case, this parameter can be omitted.

The second positional parameter indicates whether or not you want the requested file space to be contiguous. C specifies contiguous space; it is best to specify this option whenever possible because an overly fragmented file may

- Cause you to exceed the number of physical descriptors within the VTOC that are needed to describe your file's location.
- Impact performance because of excessive read/write head movement.

The third positional parameter is used to specify the number of units (tracks, cylinders) by which the file is to be dynamically extended if needed. If this parameter is omitted, the file is extended one unit at a time as needed.

The fourth positional parameter indicates the units of allocation. CYL, for example, specifies allocation by cylinder and TRK specifies allocation by track.

Track allocation is only permitted for

- All nonindexed MIRAM files
- MIRAM characteristic indexed files (see "MIRAM Concepts" under 5.1.2)

The fifth positional parameter specifies the number of units needed by this file.

The following example shows how the EXT statement is used and how it appears in the device assignment set:

```
// DVC 50
// VOL DSK001
// EXT MI,C,5,CYL,299
// LBL DMASTER
// LFD MPAY
```

In this example, we are creating a MIRAM file on a disk whose VSN is DSK001, the file identifier is DMASTER, and the name of this file in our program is MPAY. The file requires 299 cylinders and it is to be extended dynamically 5 cylinders at a time as needed.

Refer to the job control user guide for full details and formats for these statements.

Notes:

1. *In many interactive applications, the ALLOCATE command makes it unnecessary to use job control statements to allocate disk file space. For more information about this command, see the Interactive Services Operating Guide (UP-9972).*
2. *In a remote environment, you cannot create a disk file on a host system. A disk file is created by programs running on a processor that owns the supporting device.*

5.3.2. Device Assignment Set for Allocating Fixed-Head Area to a File on the 8417 Disk

The 8417 disk subsystem is available with a 4-cylinder fixed-head area. This allows faster data access because the read/write heads are fixed, rather than moveable. This feature significantly reduces the seek time required to locate records.

Determining how much fixed-head area to allocate for a MIRAM file depends on the file size and your application. You can allocate fixed-head space to hold an entire file or just the index partition of an indexed file.

To get the most from the fixed-head feature, consolidated data management controls the assignment of fixed-head area to an indexed MIRAM file as follows:

- When you request fixed-head area for an indexed file, the fixed-head area is always assigned to the file's index partition.
- Consolidated data management only assigns fixed-head area to the data partition of an indexed file if you request fixed-head area for the entire file and there is enough fixed-head area for the data partition.

When you are dealing with a large index structure, it is especially useful to allocate fixed-head space just for the coarse-level index. (A file with five keys, for example, has five separate index structures. For each structure, data management assigns a coarse-level track. These tracks make up the coarse-level index. Using the EXT statement, you can allocate fixed-head area for five tracks and data management will place the coarse-level index in these tracks.)

Note: *Remember, you can only use track allocation for MIRAM characteristic indexed files. (see "MIRAM Concepts" under 5.1.2).*

To create a file in the fixed-head area, you proceed as you would for any other disk file. You must provide a device assignment set that consists of the DVC, VOL, EXT, LBL, and LFD job control statements in that order.

Because you are dealing with the fixed-head area, the format of the DVC and EXT job control statements differ from those you would normally use when you are creating a disk file. The logical unit number in the DVC statement must be either 168 or 169, and the EXT statement must contain the FIX parameter. If this is not done, your file will not be placed in the fixed-head area.

When you use the EXT statement, the first positional parameter specifies the access method. This parameter is always MI because all disk files are MIRAM files. Because MI is the default for this parameter, it may be omitted.

The second positional parameter is C when dealing with contiguous file space.

The third positional parameter is used to specify the number of units (tracks or cylinders) by which the file is dynamically extended if needed. If this parameter is omitted, the file is extended one unit at a time as needed. Remember, file space for dynamic extension is always acquired from the moveable-head area. Your file will never have any more fixed-head space than initially allocated to it.

The fourth positional parameter is CYL when dealing with cylinders and TRK when dealing with tracks.

The fifth positional parameter specifies the number of cylinders, or tracks, needed for the file. This can be determined by using the formulas shown in 5.2.4 and 5.2.5. If you specify more than four cylinders, any data that does not fit in the 4-cylinder area will be placed in the moveable-head area.

The sixth positional parameter must always be FIX.

The following example shows how the DVC and EXT statements are used to specify that a file is to be placed in the fixed-head area and how they appear in the device assignment set:

```
// DVC 168
// VOL FDSK01
// EXT MI,C,,CYL,3,FIX
// LBL FASTAC
// LFD QDATA
```

In this example, we are creating a MIRAM file that is to be placed in the fixed-head area of an 8417 disk. The VSN is FDSK001, the file identifier is FASTAC, and the name of this file in our program is QDATA. The file requires three cylinders, and it is to be extended dynamically one cylinder at a time as needed.

If a file's device assignment set contains multiple // EXT statements requesting both fixed- and moveable-head space, consolidated data management reserves the fixed-head space for the index. When all of the fixed-head space is used, any remaining portion of the index is placed in the moveable-head area.

Refer to the *Job Control Language Programming Guide* (UP-9986) for full details and formats for these statements.

5.3.3. Device Assignment Set for Creating a Format Label Diskette File Using the Autoloader Feature of the 8420 Diskette

The 8420 diskette subsystem is available with the autoloader feature. This allows you to place several volumes in a hopper, have them processed one after the other in the order that you placed them in the hopper, and have each one automatically ejected when processing is completed on that volume. (The operator must manually feed the first volume.) This feature is useful when you are creating a multivolume sequential file because it eliminates the need to mount and remove individual volumes during the execution of your program.

When a file is opened, processing begins with the first volume and continues through each succeeding volume until the file is closed. At this point, processing is completed and the last volume is automatically ejected. (If the file is a single-volume file, the operator must eject it manually.)

A format label diskette file is treated as a disk file; consequently, you proceed as you would to create any other disk file. You must provide a device assignment set that consists of the DVC, VOL, EXT, LBL, and LFD job control statements in that order.

Because you are dealing with the autoloader feature, the logical unit number for the DVC statement must be 150 or 151. If this is not done, you cannot use the autoloader feature.

A separate VOL statement is required for each volume when you are creating a multivolume file. This is necessary because you must specify the VSN for each of the volumes that make up your file.

A separate EXT statement is required for each volume when you are creating a multivolume diskette file. This is necessary because you must allocate space on each of the volumes that make up your file.

The first positional parameter of the EXT statement is always MI. Because MI is the default for this parameter, it may be omitted.

The second positional parameter is C when you are dealing with contiguous file space.

The third positional parameter specifies the number of allocation units (cylinders, tracks) by which the volume is to be dynamically extended if needed. If this parameter is omitted, the volume is extended one unit at a time as needed.

The fourth positional parameter indicates the units of allocation. CYL, for example, specifies allocation by track.

Because you must use a separate EXT statement for each volume, the fifth positional parameter specifies the number of cylinders to be allocated to one volume in the file.

The total number of cylinders needed for a file is determined by using the formulas shown in 5.2.4 and 5.2.5.

The following example shows how the DVC, VOL, and EXT statements are used to specify that you are creating a multivolume file that is to be sequentially processed using the autoloader feature of the 8420 diskette.

```
// DVC 150
// VOL 1
// EXT MI,C,,CYL,60
// VOL 2
// EXT MI,C,,CYL,60
// VOL 3
// EXT MI,C,,CYL,55
// LBL DKAUTO
// LFD LODFIL
```

In this example, we are creating a multivolume file that is to be sequentially processed using the autoloader feature of the 8420 diskette. The file is on three volumes whose VSNs are 1, 2, and 3, respectively. The total number of cylinders required for the file is 175, and each volume is to be extended one cylinder at a time as needed. The file identifier is DKAUTO, and the name of this file in our program is LODFIL.

Refer to the *Job Control Language Programming Guide* (UP-9986) for full details and formats for these statements.

5.3.4. Device Assignment Set for an Existing Disk File

The device assignment set for an existing disk file consists of the DVC, VOL, LBL, and LFD job control statements in that order. The EXT statement is not required because the file space has already been allocated.

5.3.5. Extending an Existing Disk File

When you extend an existing disk file, the process is essentially the same as creating the file; that is, your program creates the records and writes them on the disk. The only difference is that you are dealing with an existing file rather than creating a new one. Consequently, when you execute your file extension program, you must not include an EXT statement in the device assignment set. There is also no need to specify the EXTEND parameter in the LFD statement.

5.3.6. Device Assignment Set for a Remote Disk File

A remote disk file is one that is physically associated with a processor other than the one on which your job is running.

However, both processors must run on the same operating system. The device assignment set for the remote disk file requires a // DVC job control statement to specify the host-id parameter.

The following example shows how job control statements are used to specify a remote disk file:

```
// DVC 81,HOST=AB23
// VOL D00006
// LBL FILE4
// LFD REMOTE
```

Remote file support specifications call for only

- One host per file
- External disk data files (that is, no library files)
- Single-volume disk files

Share requirements when using the host specifications include the following:

- The remote file cannot be shared among programs; that is, the ACCESS parameter specifications SADD and UCP are not permitted.
- Two jobs running on a file on a device that is physically associated with two local processors, have read-write protection if one job specifies one of the processors as host, and the other job specifies no host.

- If one of the processors is not declared host (and the disk file is physically associated with two processors), it is your responsibility to use the file in a read-only manner; that is, the ACCESS parameter specification is SRDO.

For more details about disk file sharing, see 5.4.

Note that the SIZE and VSEC parameters on the // DD control statement will be ignored if specified. For more information about remote file processing, refer to the *Job Control Language Programming Guide* (UP-9986) and the *Distributed Data Processing Programming Guide* (UP-8811).

5.3.7. Preparing Disk Volumes

All disk volumes must be prepared before data can be recorded on them. You do this by using the initialize disk routine (DSKPRP). This is described in the *System Service Programs (SSP) Operating Guide* (UP-8841).

The DSKPRP routine performs a surface analysis of the disk tracks and assigns alternate tracks if defects are discovered. It also establishes a volume table of contents on the volume so that files can be placed on it.

Note that, when you prepare a format label diskette volume, you must use a double-sided/double-density diskette.

5.4. Disk File Sharing

A data management file is a collection of related records stored on an external medium. If that medium is a disk storage device, then the individual records in the file are directly accessible. Any given reference to the file is independent of a prior reference to the file. This capability gives disk files the potential of being shared between programs. References to the file (from different programs) may be independent of one another, but they are dealing with a common set of records. If multiple programs are sharing a file and at least one of the programs is writing (adding, updating, or deleting) to the file, then this may affect the other programs that are sharing the file. It is possible for one program to read a record and take an action based on the contents of that record, and then have another program update or delete that same record. All programs that use a particular file are potential candidates to use the file at the same time (share the file), but this should only be done if the particular applications are suited to such an environment. A determination must be made for each candidate program as to what its "share requirements" are. The share requirements reflect how a program intends to use the file (read-only use or read/write use) and how other programs can concurrently use the file. The means by which you determine and specify the share requirements are discussed in detail in the succeeding paragraphs.

5.4.1. Logical Access Path (LAP)

A disk file is like any other file in that it must be assigned to a program before the program can use it. The file must be both physically and logically assigned to the program. The physical assignment is performed at job initialization, and the logical assignment is performed when the program issues a request to data management to open the file. Data management creates a logical access path (LAP) to the physical file. Associated with each LAP to a disk file are the share requirements for the particular file. The logical summation of the share requirements of all LAPs to the file defines the share environment of the disk file. When a program no longer requires the disk file, it issues a request to data management to close the file. Data management responds by removing the LAP. This *may* result in altering the share environment for the disk file.

Your sole responsibility is to determine what the LAP share requirements are. Data management automatically monitors and enforces the specified requirements. If a LAP is being created and its share requirements are not compatible with the share environment that has been established by other LAPs using the file, data management honors the requirements by suspending this program. The program is suspended at the point at which the file resource request was made and remains suspended until the LAP share requirements are compatible with the share environment.

Notes:

1. *There is a limit of 48 concurrent LAPs using a lockable disk file. When the limit is reached, an attempt by a job (or task) to gain access to the file causes the job to be suspended until the LAP count drops below 48 (due to an explicit close request or job termination).*
2. *Only one LAP can be established to a data set label diskette file at any time within a job step. If an attempt is made to open another data set or to use an already open data set, then the data management error, DM 89, occurs, indicating that the requested diskette drive is not available.*
3. *Multiple LAPs are permissible in a format label diskette within a job step. Format label diskettes are not shareable among jobs.*

5.4.2. Share Requirements

As previously mentioned, the share requirements indicate how the LAP that is being created (the current LAP) intends to access the file and the type of access that it permits other LAPs (which would be sharing the same physical file) to have. There are two ways to indicate the share requirements: the ACCESS parameter and the LFD job control statement.

- **ACCESS parameter**

If you have a BAL program, you can include an ACCESS parameter as part of the processing requirements that are specified in your program and are presented to data management when the file is opened. You can also specify the ACCESS parameter in a DD job control statement (see Appendix C) to override the program specification. If the ACCESS parameter is not specified in your program or in a DD job control statement, the default is ACCESS=EXC (see 5.4.3).

If you use IMS, you can include the ACCESS parameter for a data file in the FILE section of the configurator. You can also specify the ACCESS parameter in a DD job control statement in the IMS start-up job control stream to override the configurator specification. If the ACCESS parameter is not specified for a data file in the configurator or in a DD job control statement, the default is ACCESS=EXC.

If you have a program written in a higher-level language, the default case ACCESS=EXC always applies. If you want to specify other share requirements, you must include the ACCESS parameter in a DD job control statement.

- **LFD job control statement**

If you prefix the logical file name specified in the LFD job control statement with an asterisk (*), it indicates to data management that this file is to be a "read-only" file. This is equivalent to specifying ACCESS=SRDO (see 5.4.3). If you use this asterisk facility, it overrides the ACCESS parameter regardless of whether it was specified in your program or in a DD job control statement.

5.4.3. ACCESS Parameter Specifications

The ACCESS parameter has seven specifications for describing a file's share requirements. For efficient processing, select the ACCESS parameter specification that most accurately describes the share requirement; if you specify a greater share requirement than needed, unnecessary file share processing (I/O overhead) results. ACCESS parameter specification SADD has the highest overhead; EXCR, SRD, and UCP have moderate overhead, and EXC, SRDO, and SRDF have no overhead.

A description of each ACCESS parameter specification follows. Specifications are summarized in Table 5-3, following the detailed descriptions.

ACCESS=EXC

The LAP that declares this specification has exclusive read/write use of the file. When this specification is made, the file can only be shared with SRDF.

ACCESS=SRDO

The LAP that declares this specification is permitted to only read data from the file and it allows a number of other LAPs to share the file for read-only purposes. The SRDO specification defines a share environment for the file such that the participating LAPs sharing the file must specify SRDO, SRD, or SRDF.

ACCESS=SRDF

The LAP that declares this specification is permitted only to read data from the file. SRDF is compatible with all ACCESS parameter specifications, including EXC; however, some LAP specifications (described in the following paragraphs) may affect SRDF. Because of its compatibility with all ACCESS parameter specifications, when you use the SRDF specification to open a file, the usual compatibility checks are bypassed. Performance with SRDF is as good as with EXC and SRDO.

The following conditions affect SRDF:

- You can use SRDF for the following kinds of retrieval: UNKEYED random (relative record number) or sequential (consecutive) and KEYED retrieval. You can use KEYED retrieval as long as no other program is writing to the file and modifying the index (via record additions, or updates with key changes or key deletes). If the index is being modified when you specify SRDF, a DM24 error or an early EOF condition may result.
- If another program is updating records in the file, the program with the SRDF specification does not always receive the updated version of all records. An SRDF program does not receive updated records if records have moved into the SRDF program I/O buffer before being updated by another program. The number of records held in the SRDF buffer depends on your buffer size.
 - *For files created with the recovery option* - records added by a program that opens the file *after* the SRDF program opens the file *are not* accessible to the SRDF program; records added *before* the SRDF program opens the file *are* accessible.
 - *For files created without the recovery option* - records added by a program that has the file open at the time of the SRDF open *are not* accessible. If the program adding records closes the file (thus updating the format labels) prior to the SRDF open, then all added records *are* accessible to the SRDF program.

ACCESS=SRD

The LAP that declares this specification is permitted to only read data from the file, but it permits other LAPs to share the file for read/write purposes. The SRD is a *passive specification*. It defines a share environment that is compatible with other LAPs whose share requirements are specified on the ACCESS parameter as EXCR, SRD, SRDO, SADD, or SRDF. The EXCR, SRDO, and SADD specifications are *dominant specifications* because they define share environments that rule out specific participants.

To illustrate this, assume that a single LAP with SRD specified is currently connected to the file, and another program requests to use the same disk file and specifies its share requirements with SRDO. Because SRDO is compatible with SRD, data management honors the request and creates an SRDO specification for the LAP. This results in a share environment that excludes LAPs that require write use of the file.

Remember that the share environment is the logical summation of the share requirements of all the LAPs currently using the file. If the second request had specified EXCR instead of SRDO, the share environment would have permitted a single writer (LAP with EXCR specified) and multiple readers. If SADD had been specified instead of SRDO, the share environment would have allowed multiple writers as well as multiple readers.

Note that a record added to a file by the writer participant (LAP with EXCR or SADD specified) is immediately available to the reader participant (LAP with SRD specified), while a record just updated by the writer may not be immediately available. (It may be in a data management buffer area for a time before it's returned to the disk file.) If such a record is requested, the reader receives a valid record from the disk file but not the latest version. Also note, a writer participant can update a record just referenced by a reader participant.

ACCESS=EXCR

The LAP that declares this specification has read/write use of the file, and it allows a number of other LAPs to share the file for read-only purposes. The EXCR specification defines a single-writer/multiple-reader share environment. SRD and SRDF are compatible specifications.

ACCESS=SADD

The LAP that declares this specification has read/write use of the file and it allows other LAPs to share the file regardless of whether they require read or read/write use. The compatible specifications are SADD, SRD, and SRDF.

Physical data blocks are locked to a job when the job retrieves a record with intent to update that record or outputs a new record through a LAP with SADD specified. Locked data blocks are not available to other jobs when these locked blocks are referenced by LAPs with SADD specified. The number of blocks that are locked is dependent on the user's buffer size.

Blocks locked by input with intent to update are held until released by the followup update operation. This prevents two jobs from concurrently modifying the same physical blocks and thus avoids lost updates.

The block lock generated by an output operation is temporary and is held only for the duration of the operation. This prevents two jobs from concurrently overwriting the same physical blocks and thus avoids lost records.

A job that requests to lock blocks held by another job must wait until those blocks become available. Locked blocks, however, are always made available to jobs that request them for informational (read-only) purposes.

Only one set of blocks can be locked to a job at any one time. An attempt to accumulate block locks causes the automatic release of any block locks previously held by the job. An attempt to modify released blocks is reported as an error.

Two jobs using the same file through LAPs with SADD specified can concurrently add records to the file without losing any of the records added by either job. All added records are available to each job and all readers (LAPs with SRD specified) that may be participating.

ACCESS=UCP

The LAP that declares this specification has read/write use of the file and allows other LAPs to share the file whether they require read or read/write use. The compatible specifications are UCP, SRD, and SRDF.

With UCP specified, you accept responsibility for the data integrity of the file. This is because UCP (unlike SADD) doesn't generate block locks. This enables an application to have multiple updates pending and then to successfully update those records without receiving block lock errors (DM14 TYPE=15).

However, because block locks aren't generated, lost updates are possible in a multiwriter environment. If, for example, another job is currently updating a record in the same physical block as your job or in adjacent physical blocks, your updates could be nullified or lost. (The number of adjacent physical blocks depends on the buffer size.) Lost updates can also account for inconsistencies between fields within a record.

A DM14 TYPE=05 error message appears if you attempt to update a record that has been retrieved (with RCB) for update and subsequently deleted by another job.

If the lost updates involved a key change, the key value associated with the lost update is deleted from the index. The key value of the most recent update is reflected in the index.

Although updates can be lost, records added to a file are never lost even in a multiple adder environment.

Note: *Single-volume files are locked on physical file name and serial number; multivolume files are locked on name only. If you attempt to open a multivolume file with the same physical file name (given in the LBL job control statement), your job won't be processed. Either of the following may occur:*

- *File lockout results if the ACCESS specifications for both files are incompatible.*
- *The error message DM68 TYPE=02 results if the ACCESS specifications for both files are compatible.*

Table 5-2. Summary of ACCESS Parameter Specifications

ACCESS Parameter Specification	User Permitted	
	Current LAP	Other LAPs (and Compatible Specifications)
EXC	Read/write	Read (SRDF*)
SRDO	Read	Read (SRDO SRDF* SRD)
SRDF	Read	Read/write (EXC SRDO SRDF* SRD EXCR SADD UCP)
SRD	Read	Read/write (SRDO SRDF* EXCR SADD UCP)
EXCR	Read/write	Read (SRDF* SRD)
SADD	Read/write	Read/write (SRDF* SRD SADD)
UCP	Read/write	Read/write (SRDF* SRD UCP)

* See the SRDF description for an explanation of conditions that affect SRDF specifications.

Section 6

Data Set Label Diskette Formats and File Conventions

6.1. General Information

Data set label diskette files consist of data records that are recorded on one or more volumes (diskettes). The data records can be retrieved sequentially or randomly by relative record number (the position of a record in the file relative to the beginning of the file). The data records are recorded and retrieved via a diskette subsystem. Data set label diskette files are not shareable. For details see 5.4, "Disk File Sharing." Refer to Appendix A for the functional characteristics of the diskette subsystems that are supported.

6.2. Data Set Label Diskette File Organization

Data set label files are recorded on one or both sides of the diskette, depending on the diskette type used. The diskette type also determines the size of the fixed-length sectors on the diskette volume, the maximum number of files that the volume can contain, and the maximum number of data bytes the volume can contain. The effect of the diskette type is shown in Table 6-1.

Table 6-1. Data Set Label Diskette Characteristics

Diskette Type	Physical Sector Size in Bytes	Sectors per Track	Maximum Number of Sectors per Diskette Volume	Maximum Number of Data Bytes per Diskette Volume	Maximum Number of Files per Diskette Volume
Single Sided, Single Density*	128*	26*	1,898* or 1,924	242,944* or 246,272	19
	256	15	1,110	284,160	19
	512	8	592	303,104	19
Single Sided, Double Density	256	26	1,924	492,544	19
	512	15	1,110	568,320	19
Double Sided, Single Density	128	26	3,848	492,544	45
	256	15	2,220	568,320	45
	512	8	1,184	606,208	45
Double Sided, Double Density	256	26	3,848	985,088	45
	512	15	2,220	1,136,640	45

* Applies to files written in basic data exchange (BDE) mode - IBM System/3 and Unisys 90/30 compatibility. The number of sectors available for BDE files is reduced to have compatibility between systems. Only tracks 1 thru 73 are used in this mode. This is the only configuration available on the 8413 diskette subsystem. BDE diskettes are single sided, single density, have a logical record size ≤ 128 , fixed-length unblocked-unspanned records, and a file name of eight characters or less. Tracks 1 thru 74 are used for all other modes.

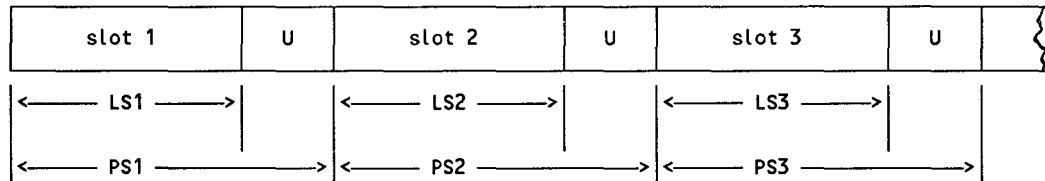
Regardless of the recording mode used, the information on a data set label diskette is organized into two areas: the index track (track 0) on which data management writes the file labels, and tracks 1 thru 74 where the data records for the file are written.

Data set label files may be either single-volume or multivolume files. In the latter case, the file can only be processed with one volume online at a time.

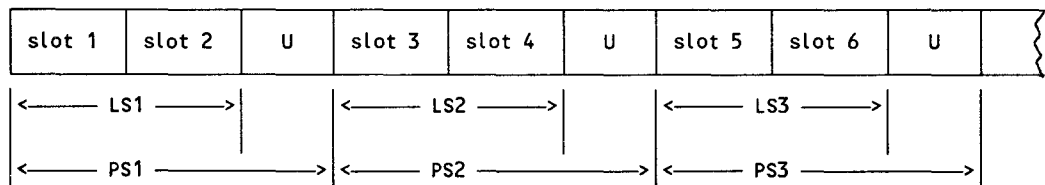
6.2.1. File Layout and Record Formats for Data Set Label Diskette Files

The file layout and the record formats for data set label diskette files are shown in Figures 6-1 and 6-2, respectively.

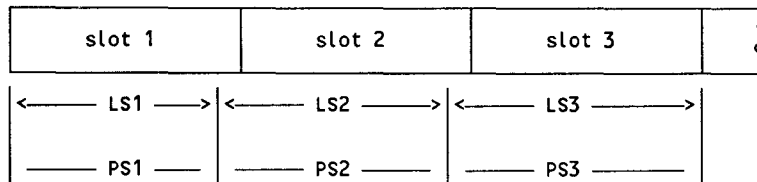
UNBLOCKED, UNSPANNED FILE



BLOCKED, UNSPANNED FILE



BLOCKED, SPANNED FILE

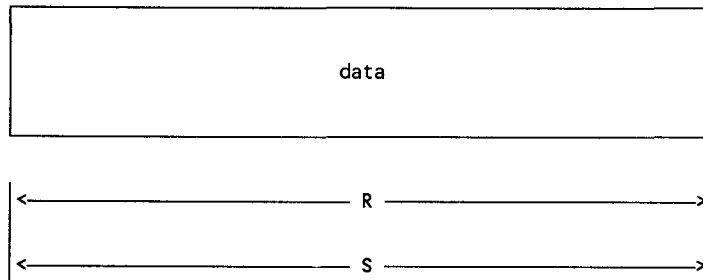


Legend

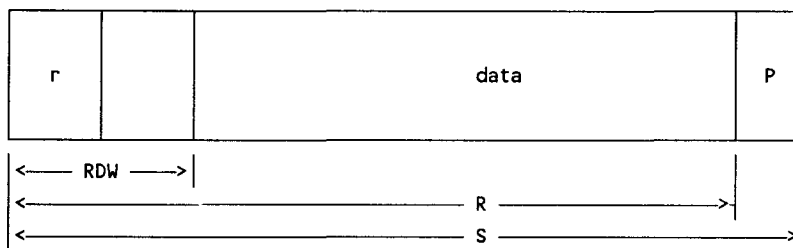
- LS Logical sector
- PS Physical sector
- U Unused portion of physical sector (PS-LS=U)

Figure 6-1. Data Set Label Diskette File Layout

FIXED-LENGTH RECORDS



VARIABLE-LENGTH RECORDS



Legend

- R Length of the physical record; record descriptor word (RDW) plus data. For variable-length records, this value, expressed in binary, must be placed in the first 2 bytes of the RDW.
- RDW 4-byte record descriptor word for variable-length records. The first 2 bytes contain the logical record length (r) expressed in binary.
- S Slot size. All records are written into fixed-size slots.
- P Padding.

Figure 6-2. Data Set Label Diskette Record Formats

6.3. Data Set Label Diskette File Job Control Considerations

A data set label diskette file is similar to a disk file in that a specific device assignment set is required in your program execution job control stream for creating a file and another is needed when you are dealing with an existing file. In addition, you also must prepare a diskette volume before you use it. These things are discussed in the paragraphs that follow.

6.3.1. Device Assignment Set for Creating a Data Set Label Diskette File

The device assignment set for creating a data set label diskette file consists of the DVC, VOL, EXT, LBL, and LFD job control statements in that order.

The DVC statement supplies the device number, the VOL statement supplies the VSN, the LBL statement supplies the file identifier, and the LFD statement associates the file name you used in your program with the diskette volume.

The EXT statement is required when you are creating a diskette file only if the area (the extent) for the file was not previously allocated via the prep routine for the data set label diskette. (The prep routine for data set label diskette automatically allocates the entire diskette for one file and assigns a file identifier of DATA unless you specify otherwise.) The extent consists of contiguous sectors on the diskette.

The first positional parameter of the EXT statement for a data set label diskette file is always MI. Because this is the default case for this statement, you can omit it if you choose.

The second positional parameter must always be C since we are dealing with contiguous sectors.

The third positional parameter must always be 0 because there is no provision for dynamic extension with data set label diskette files.

The fourth positional parameter must always be BLK because you are dealing with fixed-length sectors (blocks). This is the default case and it can be omitted if you choose.

The fifth positional parameter must be in the form (bi,ai), where bi is the block size in bytes and ai is the number of blocks in the file.

The following example shows how the EXT statement is used and how it appears in the device assignment set:

```
// DVC 130
// VOL DSKT10
// EXT MI,C,0,BLK,(80,1000),NDI
// LBL DSKTMAS
// LFD MEFILE
```

In this example, we are creating a data set label file on a diskette whose VSN is DSKT10, the file identifier is DSKTMAS, the name of this file in our program is MEFILE, and there are 1,000 sectors requested for file space on this diskette. Also, by specifying NDI, we have indicated that the file is not a basic data exchange mode file.

6.3.2. Device Assignment Set for Creating a Data Set Label Diskette File Using the Autoloader Feature of the 8420 Diskette

The 8420 diskette is available with the autoloader feature. This allows you to place several volumes in a hopper, have them processed one after the other in the order you placed them in the hopper, and have each one automatically ejected when processing is completed on that volume. (The operator must manually feed the first volume.) This feature is useful when you are creating a multivolume sequential file because it eliminates the need to mount and remove volumes during the execution of your program.

When a file is opened, processing begins with the first volume and continues through each succeeding volume until the file is closed. At this point, processing is completed and the last volume is automatically ejected. (If the file is a single-volume file, the operator must eject it manually.)

You proceed as you would to create any other data set label diskette file. You must provide a device assignment set that consists of the DVC, VOL, EXT, LBL, and LFD job control statements in that order.

Because you are dealing with the autoloader feature, the logical unit number for the DVC statement must be 150 or 151. If this is not done, you cannot use the autoloader feature.

A separate VOL statement is required when you are creating a multivolume diskette file. This is necessary because you must specify the VSN for each of the volumes that make up your file.

A separate EXT statement is required for each volume when you are creating a multivolume diskette file. This is necessary because you must allocate space on each of the volumes that make up your file.

The first positional parameter of the EXT statement for a data set label diskette file is always MI. Because MI is the default case for this parameter, it may be omitted.

The second positional parameter must always be C because we are dealing with contiguous sectors.

The third positional parameter must always be 0 because there is no provision for dynamic extension with data set label diskette files.

The fourth positional parameter must always be BLK because you are dealing with fixed-length sectors (blocks). This is the default case and it can be omitted if you choose.

The fifth positional parameter must be in the form (bi,ai), where bi is the record size in bytes and ai is the number of records.

The following example shows how the DVC, VOL, and EXT statements are used to specify that you are creating a multivolume data set label diskette file that is sequentially processed using the autoloader feature of the 8420 diskette.

```
// DVC 150
// VOL A
// EXT MI,C,0,BLK,(80,3000),NDI
// VOL B
// EXT MI,C,0,BLK,(80,3000),NDI
// VOL C
// EXT MI,C,0,BLK,(80,2560),NDI
// LBL DSLAL
// LFD OFILE
```

In this example, we are creating a multivolume data set label diskette file that is sequentially processed using the autoloader feature of the 8420 diskette. The file is on three volumes whose VSNs are A, B, and C, respectively. The data records are 80 bytes in length and the total number of records is 8,560. The file identifier is DSAL, and the name of this file in our program is OFILE. Also, by specifying NDI, we have indicated that the file is not a basic data exchange mode file.

Refer to the *Job Control Language Programming Guide* (UP-9986) for full details and formats for these statements

6.3.3. Device Assignment Set for an Existing Data Set Label Diskette File

The device assignment set for an existing data set label diskette file consists of the DVC, VOL, LBL, and LFD job control statements in that order. The EXT statement is not required because the file space has already been allocated. Refer to the *Job Control Language Programming Guide* (UP-9986) for full details and formats for these statements.

6.3.4. Preparing a Data Set Label Diskette Volume

All diskettes must be prepared before data can be recorded on them. You do this by using the initialize disk routine (DSKPRP) as described in the *System Service Programs (SSP) Operating Guide* (UP-8841).

The DSKPRP routine establishes the VTOC on the volume so that files can be written on it. This routine also establishes the recording density and that the diskette volume is to be used to record data set label files.



Section 7

Workstation Formats and File Conventions

7.1. General Information

A workstation is an I/O device that contains a keyboard and a video screen. Workstation input files consist of data records that you type in via the keyboard and output files consist of data records, created by your program, that are displayed on the video screen. Refer to Appendix A for the functional characteristics of the workstation subsystems that are supported.

Workstation files can be either single-volume or multivolume files. If a file is a single-volume file, this means that one workstation (volume) is assigned to that file. If it is a multivolume file, this means that more than one workstation is assigned to that file.

7.2. File Organization

Workstation files differ from card, tape, printer, disk, and diskette files in that data cannot be permanently stored on them. This is true because a workstation data record exists on the input or output file only as long as it appears on the screen. Once the screen is cleared, the record is gone; that is, it ceases to exist physically. As you can readily see, a workstation file is a sequential file; that is, you present your input one record at a time and your output is displayed one record at a time.

7.3. Workstation Record Formats

Workstation records consist of alphabetic, numeric, or alphanumeric data. This data must consist of displayable characters. If you include any device control characters (hexadecimal equivalent 00 through 3F), this may cause hardware errors. A record can range from one character in length to the full extent of the screen. For example, if each line on a screen can contain 80 characters and there are 24 lines on a screen, the maximum record size would be 1,920 characters.

7.4. Workstation File Job Control Considerations

The paragraphs that follow discuss the device assignment sets that you include in your program execution job control stream when you use either single-volume or multivolume workstation files.

7.4.1. Device Assignment Set for a Single-Volume Workstation File

When you use a single-volume workstation file, your device assignment set consists of a DVC and LFD job control statement. The DVC statement supplies the device number and the LFD statement associates the file name you used in your program with the workstation.

The following example shows a device assignment set for a typical single-volume workstation file:

```
// DVC 200  
// LFD DSPLY
```

In this example, a workstation is assigned to the logical file named DSPLY.

Workstation device assignment sets also require a USE job control statement whenever you use screen format services, menu services, or the dialog processor.

Refer to the *Job Control Language Programming Guide* (UP-9986) for full details and format of these statements.

7.4.2. Device Assignment Set for a Multivolume Workstation File

When you use a multivolume workstation file, it means that you intend to use more than one workstation for either input or output. Although a workstation is considered as one volume in a multivolume file, it is a separate unit. Consequently, when you have a multivolume workstation file, you must indicate how many units are assigned to the logical file in your program.

The following example shows the device assignment for a typical multivolume workstation file:

```
// DVC 200(3)  
// LFD DSPWKS
```

In this example, the file DSPWKS is a multivolume workstation file that requires three workstations. Up to 255 workstations can be assigned to any one workstation file.

Refer to the *Job Control Language Programming Guide* (UP-9986) for full details and formats of these statements.

Note: *The user must specify the maximum number of workstations per file in job control; the system does not automatically expand as more workstations are connected.*

Appendix A

Functional Characteristics of Input/Output Devices

The tables in this appendix summarize the functional characteristics of the input/output devices that are supported by consolidated data management.

Table A-1. Card Reader Subsystem Characteristics

0716 Card Reader Subsystem*	
Characteristic	Description
Card orientation (80-, 66-, and 51-column cards)	Face in, with column 1 leading and row 9 down
Card rate	1000 cpm
Read technique	Dual redundant, solar cell technique using phototransistors Column 0 amplifier checking
Read modes	Image mode: 160 six-bit characters per card Translate mode: 80 characters per card Three available codes: <ul style="list-style-type: none"> ■ 8-bit ASCII ■ 8-bit EBCDIC (required) ■ Compressed code
Read station sensing	Column by column
Hopper capacity	2400 cards
Stacker capacity Normal (stacker 2) Reject (stacker 1)	2000 cards 2000 cards

* Model 8 only

continued

Functional Characteristics of Input/Output Devices

Table A-1. Card Reader Subsystem Characteristics (cont.)

0719 Card Reader Subsystem	
Characteristic	Description
Card orientation (80-, 66-, and 51-column cards)	Face down, column 1 to left and row 9 facing away
Card rate	300 cpm
Read technique	Two columns of photosensitive sensors and light-emitting diodes Dual redundant column amplifier checking
Read modes	Image mode: 160 six-bit characters per card Translate mode: 80 characters per card
Read station sensing	Column by column
Hopper capacity	1000 cards
Stacker capacity Normal Reject	1000 cards 1000 cards

Table A-2. Card Punch Subsystem Characteristics

0608 Card Punch Subsystem	
Characteristic	Description
Media	80-column cards
Punch mode	2-column serial
Check mode	Punch motion check
Feed mode	On demand
Punch rate	75 cpm (full card) 160 cpm (28 columns only) 120 columns/second advance speed
Input capacity	700 cards
Output capacity	700 cards (primary stacker) 100 cards (auxiliary stacker)
Reading	Optional
Read rate	160 cpm

Functional Characteristics of Input/Output Devices

Table A-3. Printer Subsystem Characteristics

0770 Printer Subsystem*			
Characteristic	Description		
Print speed	0770-00	0770-02	0770-04
	112 to 1435 lpm, depending on character contingencies	213 to 2320 lpm, depending on character contingencies	337 to 3000 lpm, depending on character contingencies
	112 lpm – 384 contiguous characters	213 lpm – 384 contiguous characters	337 lpm – 384 contiguous characters
	800 lpm – 48 contiguous characters	1400 lpm – 48 contiguous characters	2000 lpm – 48 contiguous characters
	1435 lpm – 24 contiguous characters	2320 lpm – 24 contiguous characters	3000 lpm – 24 contiguous characters
Line advance rate	50 ips	75 ips	100 ips
Line advance timing	Advance and print	Time (ms)	
		6 lpi	8 lpi
	1 line 2 lines 3 lines n + 1 line	120.0 127.6 135.2 120+7.6n	118.0 123.7 129.4 118+5.7n
Number of print positions	Full print width of 132 print positions placed anywhere on a 16.5-inch form. With 22-inch form, only central 13.2-inch portion can be used (160 print positions with feature).		
Forms advance control	Vertical format buffer		
Form dimensions	Continuous forms with standard edge sprocket holes from 4 to 22 inches in width. Carbons may be attached or unattached with multicopy forms to a maximum of six parts. Recommended pack thickness not to exceed .0155 inch for high quality print.		
Character set	Standard 48-character set. Any number of characters up to 384 with options.		
Horizontal spacing	10 characters per inch		
Vertical spacing	6 or 8 lines per inch, as determined by program		

* Model 8 only

continued

Functional Characteristics of Input/Output Devices

Table A-3. Printer Subsystem Characteristics (cont.)

0776 Printer Subsystem			
Characteristic	Description		
Print speed	210 to 1250 lpm depending on character contingencies:		
	Available character sets (characters/set)	Number of sets per band	Nominal print rate (lpm)
	384	1	210
	192	2	395
	128	3	560
	96	4	710
	64	6	980
	48	8	1200
Line advance timing	Advance and print (number of lines)	Time (ms)	
		6 lpi	8 lpi
	1	16.7	14.1
	2	24.6	20.9
	3	30.9	25.9
	4	35.0	30.9
	5	38.9	34.1
	6	42.6	37.1
	7	45.9	39.9
	8	49.3	42.6
Number of print positions	136 print positions (columns)		
Forms advance control	Vertical format buffer		
Forms advance rate	50 inches (127 cm) per second		
Form dimensions	4 to 18.75 inches (10.16 to 47.62 cm) wide 1 to 18 inches (45.72 cm) long		
Horizontal spacing	10 characters per inch		
Vertical spacing	6 or 8 lines per inch, operator-selectable		

continued

Functional Characteristics of Input/Output Devices

Table A-3. Printer Subsystem Characteristics (cont.)

0789 Printer Subsystem			
Characteristic	Description		
Print speed	180, 300, and 640 lpm, depending on character contingencies		
	Available character sets	Number of sets per band	Nominal print rate (lpm)
	48	4 (plus 16 char)	317
	64	3 (plus 16 char)	306
	96	2 (plus 16 char)	246
	128	1 (plus 80 char)	177
Line advance timing	Advance and print	Time (ms)	
		6 lpi	8 lpi
	1 line	40	40
	2 lines	52	52
	3 lines	64	64
n+1 lines	76+12	76+12	
Number of print positions	120 print positions (columns) by standard printer; 132 columns by feature		
Forms advance control	Controlled from host processor		
Forms advance rate	15 inches (38.1 cm) per second		
Form dimensions	3 to 15 inches (7.62 to 38.10 cm) wide 1 to 22 inches (55.88 cm) long		
Horizontal spacing	10 characters per inch		
Vertical spacing	6 or 8 lines per inch, operator-selectable		

Table A-4. Disk and Diskette Subsystem Characteristics

Characteristics	Description								
	8416 Disk Subsystem ^①	8417 Disk Subsystem	8418 Disk Subsystem ^①	8419 Disk Subsystem	8420/8422 Diskette Subsystem	8430 Disk Subsystem ^①	8433 Disk Subsystem ^①	8470 Disk Subsystem	8494 Disk Subsystem
Data capacity MB (8-bit bytes) formatted	28.95	118.2	28.9 or 57.9	72.39	Single density ^④ 1 side 303,104 ^② 2 sides 606,208 Double density ^④ 1 side 563,320 2 sides 1,136,640 ^③	100	200	491	308
Number of disk units per subsystem	2 to 8	1 to 8	2 to 8	1 to 8	1 to 4	1 to 8 (with optional feature up to 16)	1 to 8 (with optional feature up to 16)	1 to 8	2 to 8 ^⑦
Disk/diskette speed (rpm)	2800	3400	2800	2800	360	3600	3600	3600	3600
Rotation period (ms/rotation)	21.5	17.6	21.5	21	166	16.7	16.7	16.7	16.7
Data bit rate (MHz)	5.0	9.05	5.0	6.2	-	6.45	6.45	16.8	14.5
Bit density (ppi)	4040	6366	4040	5050	-	4040	4040	11,134	15,000
Track density (tracks/inch)	192	476	370	-	-	192	370	630	960
Track capacity (bytes/track) formatted	10,240 ^⑤	15,360	10,240	12,800	3328 to 7680 ^⑤	13,030	13,030	24,576	24,576
Number of cylinders	404 + 7 alternates	550 + 10 alternates	404 or 808 + 7 alternates	808 + 7 alternates	77 total, 75 ^⑥ for data use per diskette surface	404 + 7 alternates	808 + 7 alternates	625 + 5 alternates	626 + 6 ^⑥
Number of surfaces per disk unit	Data 7 positioning 1	14	Data 7 positioning 1	7	2	19	19	32	20
Positioning time (seek time)	Minimum (ms) Average (ms) Maximum (ms)	7 35 70	10 27 45	10 33 60	3 15 35	7 27 50	10 30 55	4 23 46	5 18 35
Transfer rate (kilobytes/second)	625	1130	628	784	Dependent on sector sequence arrangement	806	806	2,100	2,200

Notes:

- ① Model 8 only
- ② 242,944 for data set label BDE (basic data exchange) diskette file.
- ③ 971,776 for format label diskette file.
- ④ Maximum value. Actual value is dependent on diskette type (single sided, single density; single sided, double density; double sided, single density; double sided, double density), physical sector size (128, 256, or 512 bytes) and file type (format label or data set label).
- ⑤ In fixed 256-byte sectors, 40 sectors per track
- ⑥ 73 for format label diskette file and data set label BDE (basic data exchange) file. 75 for other data set label non-BDE files.
- ⑦ Only increments of 2 (2, 4, 6, 8) are supported.
- ⑧ Equivalent of approximately six cylinders of available alternate sectors.

Table A-5. Magnetic Tape Subsystem Characteristics

Characteristic	Description																
	UNISERVO 12*		UNISERVO 16*		UNISERVO 20*		UNISERVO 10*		UNISERVO 14*		UNISERVO 10		UNISERVO 22*		UNISERVO 24*		Streaming Tape Drive†
Tape units per subsystem	1 to 16		1 to 16		1 to 16		2 to 8		2 to 8		1 to 8		1 to 8		1 to 8		1 to 4 daisy chained to host controller
Data transfer rate (maximum) (frames per second)	68,320		192,000		320,000		40,000		96,000		40,000		120,000		200,000		160,000 or 40,000
Tape speed (inches per second)	42.7		120		200		25		60		25		75		125		100 or 25
Tape direction Reading Writing	Forward or backward Forward		Forward or backward Forward		Forward or backward Forward		Forward or backward Forward		Forward or backward Forward		Forward or backward Forward		Forward or backward Forward		Forward or backward Forward		Forward or backward Forward
Tape length (maximum) (feet)	2400		2400		2400		2400		2400		2400		2400		2400		2400
Tape thickness (mils)	1.5		1.5		1.5		1.5		1.5		1.5		1.5		1.5		1.5
Block length	Variable		Variable		Variable		Variable		Variable		Variable		Variable		Variable		Variable
Maximum block size (bytes)	65,535		65,535		65,535		8191		65,535		65,535		65,535		65,535		65,535
Minimum block size (bytes)	18		18		18		18		18		18		18		18		18
Interblock gap (inches)	9-track 0.6	7-track 0.75	9-track 0.6	7-track 0.75	0.6		9-track 0.6	7-track 0.75	9-track 0.6	7-track 0.75	0.6		0.6	0.6	0.6		0.6
Interblock gap time (ms) Nonstop Start-stop	14.1 20.1	17.6 23.6	5.0 8.0	6.25 9.25	3.0 5.0		24 30	30 38	14 20	17 23	24		8.0 7.7	4.8 7.7	6.0 7.7		
Pulse density (ppi)	1600 800	800 556 200	1600 800	800	1600		1600 800	800 556 200	1600 800	800 556	1600 on 9-track PE 800 on 9-track NRZI		1600 or 800 on 9-track PE/NRZI		1600 or 800 on 9-track PE/NRZI		1600 on 9-track PE
Recording mode	PE	NRZI	PE	NRZI	PE		PE or NRZI	NRZI	PE or NRZI	NRZI	PE or NRZI		PE or NRZI	PE or NRZI	PE		
Reversal time (ms)	25		10		16		16		10		16		10	10	450		
Rewind time (min)	3		2		1		3		3		3		2	2	3		
Simultaneous operation	Optional		Optional		Optional		Optional		Optional		Optional		Optional	Optional	Optional		

* Model 8 only

† Provides dump or restore capability to backup data stored on nonremovable disk packs.

Functional Characteristics of Input/Output Devices

Table A-6. Workstation Subsystem Characteristics

Workstation Subsystem	
Characteristic	Description
Type of display	Cathode ray tube (CRT)
Number of display lines	24 plus 1 indicator
Characters per line	80
Number of units	1 to 8
Keyboard arrangements	Typewriter layout, typewriter layout with numeric and function pads, or Katakana/English
Character sets	Domestic, United Kingdom, Germany, France, Spain, Denmark/Norway, Sweden/Finland, Italy, or Katakana/English

Appendix B

Code Correspondences

B.1. General Information

This appendix presents a cross-reference table and several figures useful to you in visualizing the correspondences among the following codes commonly used in data processing:

- Hollerith punched card code
- EBCDIC (Extended Binary-Coded Decimal Interchange Code)
- ASCII (American National Standard Code for Information Interchange)
- Binary bit pattern (bit configuration) representation for an 8-bit system.
- Hexadecimal representation
- Compressed code for punched cards
- Binary (image) mode for punched cards

B.2. EBCDIC/ASCII/Hollerith Correspondence

Table B-1 cross-references the correspondences between the Hollerith punched card code, ASCII, and EBCDIC. The table is arranged in the sorting (or collating) sequence of the binary bit patterns that have been assigned to the codes, with 0000 0000 being the lowest value in the sequence and 1111 1111 the highest.

Note that the column headed Decimal uses decimal numbers to represent the positions of the codes and bit patterns in this sequence, but counts the position of the lowest value as the 0th (zeroth) position rather than the first. Thus, the position of the highest value bit pattern 1111 1111 is represented in the decimal column by 255, whereas it is actually the 256th in the sequence. This scheme corresponds to the common convention for numbering bytes, in which the first byte of a group is byte 0, and is convenient when you are constructing a 256-byte translation table.

The column headed Decimal also represents the collating sequence for the EBCDIC graphic characters shown in the fourth column of the table; the fifth column, Hollerith Punched Card Code, contains the hole patterns assigned to these EBCDIC graphics. An empty space in the fourth column represents the position of the EBCDIC control character for which there is no graphic representation; the EBCDIC space character is

represented in the fourth column by the conventional notation SP at decimal position 64, and the corresponding card code is "no punches."

The ASCII graphic characters, listed in the sixth column of Table B-1, are also in their collating sequence, and the hole patterns in the seventh column correspond to the ASCII graphics. The ASCII space character is represented by the notation SP in the sixth column at decimal position 32; the corresponding card code is, again, "no punches." The empty space in the sixth column represents the positions of the ASCII control characters for which there is no graphic representation. The shading in the ASCII graphic character column indicates where the 128-character ASCII code leaves off; there are no ASCII graphic or control characters that correspond to the bit patterns higher in collating sequence than 0111 1111 (the 128th in Table B-1).

B.2.1. Hollerith Punched Card Code

The standard Hollerith punched card code specifies 256 unique hole patterns in 12-row punched cards. Hole patterns are assigned to the 128 characters of ASCII and to 128 additional characters for use in 8-bit coded systems. These include the EBCDIC set. Note that no sorting sequence is implied by the Hollerith code itself.

B.2.2. EBCDIC

EBCDIC is an extension of Hollerith coding practices. It comprises 256 characters, each of which is represented by an 8-bit pattern. Table B-1 shows the EBCDIC graphic characters only; the EBCDIC control characters are not indicated.

B.2.3. ASCII

ASCII comprises 128 coded characters, each represented by an 8-bit pattern, and includes both control characters and graphic characters. Only the latter are shown in Table B-1. ASCII is used for information interchange among data processing communication systems and associated equipment.

Table B-1. Cross-Reference Table: EBCDIC/ASCII/Hollerith

EBCDIC					ASCII	
Decimal	Hexadecimal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
0	00	0000 0000		12-0-9-8-1		12-0-9-8-1
1	01	0000 0001		12-9-1		12-9-1
2	02	0000 0010		12-9-2		12-9-2
3	03	0000 0011		12-9-3		12-9-3
4	04	0000 0100		12-9-4		9-7
5	05	0000 0101		12-9-5		0-9-8-5
6	06	0000 0110		12-9-6		0-9-8-6
7	07	0000 0111		12-9-7		0-9-8-7
8	08	0000 1000		12-9-8		11-9-6
9	09	0000 1001		12-9-8-1		12-9-5
10	0A	0000 1010		12-9-8-2		0-9-5
11	0B	0000 1011		12-9-8-3		12-9-8-3
12	0C	0000 1100		12-9-8-4		12-9-8-4
13	0D	0000 1101		12-9-8-5		12-9-8-5
14	0E	0000 1110		12-9-8-6		12-9-8-6
15	0F	0000 1111		12-9-8-7		12-9-8-7
16	10	0001 0000		12-11-9-8-1		12-11-9-8-1
17	11	0001 0001		11-9-1		11-9-1
18	12	0001 0010		11-9-2		11-9-2
19	13	0001 0011		11-9-3		11-9-3
20	14	0001 0100		11-9-4		9-8-4
21	15	0001 0101		11-9-5		9-8-5
22	16	0001 0110		11-9-6		9-2
23	17	0001 0111		11-9-7		0-9-6
24	18	0001 1000		11-9-8		11-9-8
25	19	0001 1001		11-9-8-1		11-9-8-1
26	1A	0001 1010		11-9-8-2		9-8-7
27	1B	0001 1011		11-9-8-3		0-9-7
28	1C	0001 1100		11-9-8-4		11-9-8-4
29	1D	0001 1101		11-9-8-5		11-9-8-5
30	1E	0001 1110		11-9-8-6		11-9-8-6
31	1F	0001 1111		11-9-8-7		11-9-8-7
32	20	0010 0000		11-0-9-8-1	SP	No punches
33	21	0010 0001		0-9-1	!	12-8-7
34	22	0010 0010		0-9-2	"	8-7
35	23	0010 0011		0-9-3	#	8-3
36	24	0010 0100		0-9-4	\$	11-8-3
37	25	0010 0101		0-9-5	%	0-8-4
38	26	0010 0110		0-9-6	&	12
39	27	0010 0111		0-9-7	'	8-5
40	28	0010 1000		0-9-8	(12-8-5
41	29	0010 1001		0-9-8-1)	11-8-5
42	2A	0010 1010		0-9-8-2	*	11-8-4
43	2B	0010 1011		0-9-8-3	+	12-8-6
44	2C	0010 1100		0-9-8-4	,	0-8-3
45	2D	0010 1101		0-9-8-5	-	11
46	2E	0010 1110		0-9-8-6	.	12-8-3
47	2F	0010 1111		0-9-8-7	/	0-1
48	30	0011 0000		12-11-0-9-8-1	0	0
49	31	0011 0001		9-1	1	1
50	32	0011 0010		9-2	2	2
51	33	0011 0011		9-3	3	3
52	34	0011 0100		9-4	4	4
53	35	0011 0101		9-5	5	5
54	36	0011 0110		9-6	6	6

continued

Code Correspondences

Table B-1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (cont.)

EBCDIC					ASCII	
Decimal	Hexadecimal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
55	37	0011 0111		9-7	7	7
56	38	0011 1000		9-8	8	8
57	39	0011 1001		9-8-1	9	9
58	3A	0011 1010		9-8-2	:	8-2
59	3B	0011 1011		9-8-3	;	11-8-6
60	3C	0011 1100		9-8-4	<	12-8-4
61	3D	0011 1101		9-8-5	=	8-6
62	3E	0011 1110		9-8-6	>	0-8-6
63	3F	0011 1111		9-8-7	?	0-8-7
64	40	0100 0000	SP	No punches	@	8-4
65	41	0100 0001		12-0-9-1	A	12-1
66	42	0100 0010		12-0-9-2	B	12-2
67	43	0100 0011		12-0-9-3	C	12-3
68	44	0100 0100		12-0-9-4	D	12-4
69	45	0100 0101		12-0-9-5	E	12-5
70	46	0100 0110		12-0-9-6	F	12-6
71	47	0100 0111		12-0-9-7	G	12-7
72	48	0100 1000		12-0-9-8	H	12-8
73	49	0100 1001		12-8-1	I	12-9
74	4A	0100 1010	[12-8-2	J	11-1
75	4B	0100 1011	.	12-8-3	K	11-2
76	4C	0100 1100	<	12-8-4	L	11-3
77	4D	0100 1101	(12-8-5	M	11-4
78	4E	0100 1110	+	12-8-6	N	11-5
79	4F	0100 1111	!	12-8-7	O	11-6
80	50	0101 0000	&	12	P	11-7
81	51	0101 0001		12-11-9-1	Q	11-8
82	52	0101 0010		12-11-9-2	R	11-9
83	53	0101 0011		12-11-9-3	S	0-2
84	54	0101 0100		12-11-9-4	T	0-3
85	55	0101 0101		12-11-9-5	U	0-4
86	56	0101 0110		12-11-9-6	V	0-5
87	57	0101 0111		12-11-9-7	W	0-6
88	58	0101 1000		12-11-9-8	X	0-7
89	59	0101 1001		11-8-1	Y	0-8
90	5A	0101 1010]	11-8-2	Z	0-9
91	5B	0101 1011	\$	11-8-3	[12-8-2
92	5C	0101 1100	*	11-8-4	\	0-8-2
93	5D	0101 1101	}	11-8-5]	11-8-2
94	5E	0101 1110	;	11-8-6	^	11-8-7
95	5F	0101 1111	^	11-8-7	~	0-8-5
96	60	0110 0000	-	11	`	8-1
97	61	0110 0001	/	0-1	a	12-0-1
98	62	0110 0010		11-0-9-2	b	12-0-2
99	63	0110 0011		11-0-9-3	c	12-0-3
100	64	0110 0100		11-0-9-4	d	12-0-4
101	65	0110 0101		11-0-9-5	e	12-0-5
102	66	0110 0110		11-0-9-6	f	12-0-6
103	67	0110 0111		11-0-9-7	g	12-0-7
104	68	0110 1000		11-0-9-8	h	12-0-8
105	69	0110 1001	:	0-8-1	i	12-0-9
106	6A	0110 1010	:	12-11	j	12-11-1
107	6B	0110 1011	,	0-8-3	k	12-11-2
108	6C	0110 1100	%	0-8-4	l	12-11-3
109	6D	0110 1101	—	0-8-5	m	12-11-4

continued

Table B-1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (cont.)

EBCDIC					ASCII	
Decimal	Hexadecimal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
110	6E	0110 1110	>	0-8-6	n	12-11-5
111	6F	0110 1111	?	0-8-7	o	12-11-6
112	70	0111 0000		12-11-0	p	12-11-7
113	71	0111 0001		12-11-0-9-1	q	12-11-8
114	72	0111 0010		12-11-0-9-2	r	12-11-9
115	73	0111 0011		12-11-0-9-3	s	11-0-2
116	74	0111 0100		12-11-0-9-4	t	11-0-3
117	75	0111 0101		12-11-0-9-5	u	11-0-4
118	76	0111 0110		12-11-0-9-6	v	11-0-5
119	77	0111 0111		12-11-0-9-7	w	11-0-6
120	78	0111 1000	,	12-11-0-9-8	x	11-0-7
121	79	0111 1001		8-1	y	11-0-8
122	7A	0111 1010	:	8-2	z	11-0-9
123	7B	0111 1011	#	8-3	z	12-0
124	7C	0111 1100	@	8-4	z	12-11
125	7D	0111 1101	.	8-5	z	11-0
126	7E	0111 1110	=	8-6	z	11-0-1
127	7F	0111 1111	"	8-7	z	12-9-7
128	80	1000 0000		12-0-8-1	z	11-0-9-8-1
129	81	1000 0001	a	12-0-1		0-9-1
130	82	1000 0010	b	12-0-2		0-9-2
131	83	1000 0011	c	12-0-3		0-9-3
132	84	1000 0100	d	12-0-4		0-9-4
133	85	1000 0101	e	12-0-5		11-9-5
134	86	1000 0110	f	12-0-6		12-9-6
135	87	1000 0111	g	12-0-7		11-9-7
136	88	1000 1000	h	12-0-8		0-9-8
137	89	1000 1001	i	12-0-9		0-9-8-1
138	8A	1000 1010		12-0-8-2		0-9-8-2
139	8B	1000 1011		12-0-8-3		0-9-8-3
140	8C	1000 1100		12-0-8-4		0-9-8-4
141	8D	1000 1101		12-0-8-5		12-9-8-1
142	8E	1000 1110		12-0-8-6		12-9-8-2
143	8F	1000 1111		12-0-8-7		11-9-8-3
144	90	1001 0000		12-11-8-1		12-11-0-9-8-1
145	91	1001 0001	j	12-11-1		9-1
146	92	1001 0010	k	12-11-2		11-9-8-2
147	93	1001 0011	l	12-11-3		9-3
148	94	1001 0100	m	12-11-4		9-4
149	95	1001 0101	n	12-11-5		9-5
150	96	1001 0110	o	12-11-6		9-6
151	97	1001 0111	p	12-11-7		12-9-8
152	98	1001 1000	q	12-11-8		9-8
153	99	1001 1001	r	12-11-9		9-8-1
154	9A	1001 1010		12-11-8-2		9-8-2
155	9B	1001 1011		12-11-8-3		9-8-3
156	9C	1001 1100		12-11-8-4		12-9-4
157	9D	1001 1101		12-11-8-5		11-9-4
158	9E	1001 1110		12-11-8-6		9-8-6
159	9F	1001 1111		12-11-8-7		11-0-9-1

continued

Code Correspondences

Table B-1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (cont.)

EBCDIC					ASCII	
Decimal	Hexadecimal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
160	A0	1010 0000		11-0-8-1		12-0-9-1
161	A1	1010 0001	~	11-0-1		12-0-9-2
162	A2	1010 0010	s	11-0-2		12-0-9-3
163	A3	1010 0011	t	11-0-3		12-0-9-4
164	A4	1010 0100	u	11-0-4		12-0-9-5
165	A5	1010 0101	v	11-0-5		12-0-9-6
166	A6	1010 0110	w	11-0-6		12-0-9-7
167	A7	1010 0111	x	11-0-7		12-0-9-8
168	A8	1010 1000	y	11-0-8		12-8-1
169	A9	1010 1001	z	11-0-9		12-11-9-1
170	AA	1010 1010		11-0-8-2		12-11-9-2
171	AB	1010 1011		11-0-8-3		12-11-9-3
172	AC	1010 1100		11-0-8-4		12-11-9-4
173	AD	1010 1101		11-0-8-5		12-11-9-5
174	AE	1010 1110		11-0-8-6		12-11-9-6
175	AF	1010 1111		11-0-8-7		12-11-9-7
176	B0	1011 0000		12-11-0-8-1		12-11-9-8
177	B1	1011 0001		12-11-0-1		11-8-1
178	B2	1011 0010		12-11-0-2		11-0-9-2
179	B3	1011 0011		12-11-0-3		11-0-9-3
180	B4	1011 0100		12-11-0-4		11-0-9-4
181	B5	1011 0101		12-11-0-5		11-0-9-5
182	B6	1011 0110		12-11-0-6		11-0-9-6
183	B7	1011 0111		12-11-0-7		11-0-9-7
184	B8	1011 1000		12-11-0-8		11-0-9-8
185	B9	1011 1001		12-11-0-9		0-8-1
186	BA	1011 1010		12-11-0-8-2		12-11-0
187	BB	1011 1011		12-11-0-8-3		12-11-0-9-1
188	BC	1011 1100		12-11-0-8-4		12-11-0-9-2
189	BD	1011 1101		12-11-0-8-5		12-11-0-9-3
190	BE	1011 1110		12-11-0-8-6		12-11-0-9-4
191	BF	1011 1111		12-11-0-8-7		12-11-0-9-5
192	C0	1100 0000	{	12-0		12-11-0-9-6
193	C1	1100 0001	A	12-1		12-11-0-9-7
194	C2	1100 0010	B	12-2		12-11-0-9-8
195	C3	1100 0011	C	12-3		12-0-8-1
196	C4	1100 0100	D	12-4		12-0-8-2
197	C5	1100 0101	E	12-5		12-0-8-3
198	C6	1100 0110	F	12-6		12-0-8-4
199	C7	1100 0111	G	12-7		12-0-8-5
200	C8	1100 1000	H	12-8		12-0-8-6
201	C9	1100 1001	I	12-9		12-0-8-7
202	CA	1100 1010		12-0-9-8-2		12-11-8-1
203	CB	1100 1011		12-0-9-8-3		12-11-8-2
204	CC	1100 1100		12-0-9-8-4		12-11-8-3
205	CD	1100 1101		12-0-9-8-5		12-11-8-4
206	CE	1100 1110		12-0-9-8-6		12-11-8-5
207	CF	1100 1111		12-0-9-8-7		12-11-8-6
208	D0	1101 0000	}	11-0		12-11-8-7
209	D1	1101 0001	J	11-1		11-0-8-1

continued

Table B-1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (cont.)

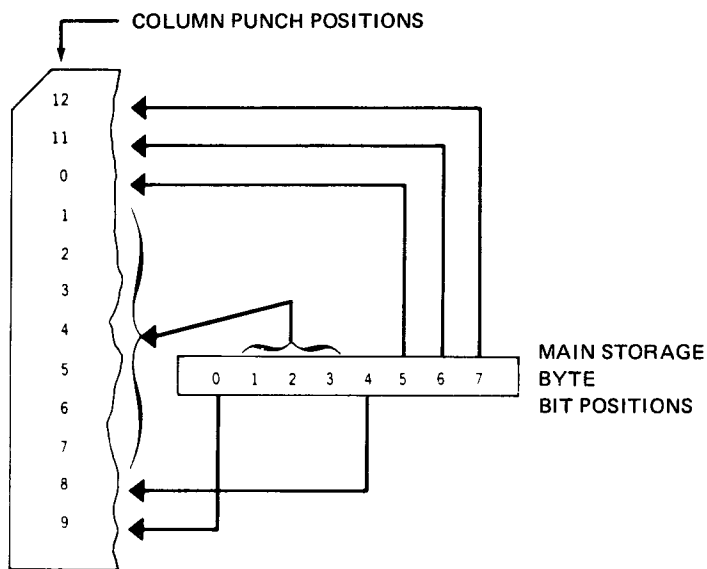
EBCDIC					ASCII	
Decimal	Hexadecimal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
210	D2	1101 0010	K	11-2		11-0-8-2
211	D3	1101 0011	L	11-3		11-0-8-3
212	D4	1101 0100	M	11-4		11-0-8-4
213	D5	1101 0101	N	11-5		11-0-8-5
214	D6	1101 0110	O	11-6		11-0-8-6
215	D7	1101 0111	P	11-7		11-0-8-7
216	D8	1101 1000	Q	11-8		12-11-0-8-1
217	D9	1101 1001	R	11-9		12-11-0-1
218	DA	1101 1010		12-11-9-8-2		12-11-0-2
219	DB	1101 1011		12-11-9-8-3		12-11-0-3
220	DC	1101 1100		12-11-9-8-4		12-11-0-4
221	DD	1101 1101		12-11-9-8-5		12-11-0-5
222	DE	1101 1110		12-11-9-8-6		12-11-0-6
223	DF	1101 1111		12-11-9-8-7		12-11-0-7
224	E0	1110 0000	\	0-8-2		12-11-0-8
225	E1	1110 0001		11-0-9-1		12-11-0-9
226	E2	1110 0010	S	0-2		12-11-0-8-2
227	E3	1110 0011	T	0-3		12-11-0-8-3
228	E4	1110 0100	U	0-4		12-11-0-8-4
229	E5	1110 0101	V	0-5		12-11-0-8-5
230	E6	1110 0110	W	0-6		12-11-0-8-6
231	E7	1110 0111	X	0-7		12-11-0-8-7
232	E8	1110 1000	Y	0-8		12-0-9-8-2
233	E9	1110 1001	Z	0-9		12-0-9-8-3
234	EA	1110 1010		11-0-9-8-2		12-0-9-8-4
235	EB	1110 1011		11-0-9-8-3		12-0-9-8-5
236	EC	1110 1100		11-0-9-8-4		12-0-9-8-6
237	ED	1110 1101		11-0-9-8-5		12-0-9-8-7
238	EE	1110 1110		11-0-9-8-6		12-11-9-8-2
239	EF	1110 1111		11-0-9-8-7		12-11-9-8-3
240	F0	1111 0000	0	0		12-11-9-8-4
241	F1	1111 0001	1	1		12-11-9-8-5
242	F2	1111 0010	2	2		12-11-9-8-6
243	F3	1111 0011	3	3		12-11-9-8-7
244	F4	1111 0100	4	4		11-0-9-8-2
245	F5	1111 0101	5	5		11-0-9-8-3
246	F6	1111 0110	6	6		11-0-9-8-4
247	F7	1111 0111	7	7		11-0-9-8-5
248	F8	1111 1000	8	8		11-0-9-8-6
249	F9	1111 1001	9	9		11-0-9-8-7
250	FA	1111 1010		12-11-0-9-8-2		12-11-0-9-8-2
251	FB	1111 1011		12-11-0-9-8-3		12-11-0-9-8-3
252	FC	1111 1100		12-11-0-9-8-4		12-11-0-9-8-4
253	FD	1111 1101		12-11-0-9-8-5		12-11-0-9-8-5
254	FE	1111 1110		12-11-0-9-8-6		12-11-0-9-8-6
255	FF	1111 1111		12-11-0-9-8-7		12-11-0-9-8-7

B.3. Other Card Codes

There are three other card coding systems that can be handled by data management: compressed code, column binary (image) code, and 96-column code.

B.3.1. Compressed Card Code

Figure B-1 indicates the construction of the compressed card code; each card column is represented by an 8-bit pattern in 1 byte of main storage.



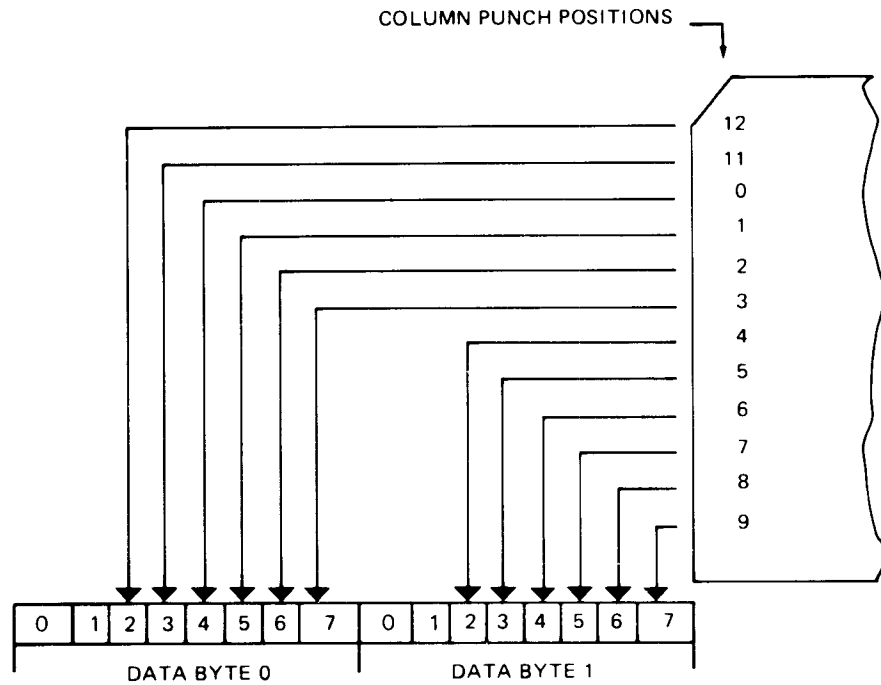
Note: Punch positions 1 through 7 are indicated in bits 1 through 3, according to the following table:

Punch Rows 1 Thru 7	Bits 123
NONE	000
1	011
2	101
3	001
4	010
5	100
6	111
7	110

Figure B-1. Compressed Card Code

B.3.2. Column Binary (Image) Code

Figure B-2 indicates the construction of this code. Note that each card column requires two bytes of main storage; an I/O area of 160 bytes is required for an 80-column card.



Note: Bits 0 and 1 are cleared to zeros on an image read.

Figure B-2. Column Binary (Image) Card Code



Appendix C

DD Job Control Statement Processing

C.1. General Information

The DD job control statement can be used in your program execution job control stream to temporarily change certain file parameters at run time. The changes are effective only during the execution of the job. If a permanent change is desired, you must change your source program. The *Job Control Language Programming Guide* (UP-9986) describes the format and placement of the DD job control statement.

C.2. DD Job Control Statement Parameters

Table C-1 summarizes the DD job control statement keyword parameters you can use to temporarily change your file parameters. If you specify a keyword parameter that is not allowed for the particular type of file, the specification is ignored.

Table C-1. Allowable Keyword Parameters for the DD Job Control Statement

Keyword	Format Label Diskette/Disk	Data Set Label Diskette	Tape	Card	Printer
RCFM*			X	X	X
BKSZ*	X	X	X	X	X
RCSZ*	X	X	X		
KLENI-5*	X				
KLOC1-5*	X				
INDS*	X				
SIZE	X				
ACCESS	X				
VSEC	V				
RECV	X				
VMNT	X	X			

continued

Table C-1. Allowable Keyword Parameters for the DD Job Control Statement (cont.)

Keyword	Format Label Diskette/Disk	Data Set Label Diskette	Tape	Card	Printer
RCB	X	X			
OFFSET		X			
REWIND			X		
OPRW			X		
CLRW			X		
FILABL			X		
TPMARK			X		
RESTORE	X				
CACHE	X				
MSGSUPP	X				

* Be careful when specifying this keyword parameter. If the program accessing the file is dependent on predefined (e.g., compile time) file or processing characteristics, it may not be prepared for such a change at execution time. You may get unexpected results unless the program is a user-written BAL program prepared for this type of specification change or if the user documentation for the product explicitly states that this specification can be changed at execution time.

Legend

- X Allowable keyword
- V Applies to nonsectorized disk devices only

C.2.1. Record Format (RCFM)

This keyword parameter specifies the record format.

RCFM=FIXUNB

Specifies fixed-length unblocked records.

RCFM=FIXBLK

Specifies fixed-length blocked records. This specification is invalid for card and printer files.

RCFM=UNDEF
Specifies undefined records.

RCFM=VARUNB
Specifies variable-length unblocked records.

RCFM=VARBLK
Specifies variable-length blocked records. This specification is invalid for card and printer files.

C.2.2. Data Buffer/Block Size (BKSZ)

This keyword parameter specifies the size of the data I/O buffer.

BKSZ=n
Specifies the size of the buffer in bytes. If the full size specified cannot be used, it will be automatically rounded down to an appropriate size.

The following is device-dependent information regarding block/buffer size:

DISK/FORMAT LABEL DISKETTE/DATA SET LABEL DISKETTE

The following algorithm determines the minimum allowable buffer size in logical sectors

$$S/LSS=N \text{ and } R$$

where:

S
Is the slot size. The slot size equals the record size + 1 for files with an RCB and fixed-length records. Otherwise, the slot size equals the record size.

LSS
Is the logical sector size. For disk and format label diskette files, the logical sector size is equal to the physical sector size. For the data set label diskette, the logical sector size is:

- equal to the record size if records are unblocked and unspanned.
- $n \times$ record size if records are blocked and unspanned.
- physical sector size if records are blocked and spanned.

N
Is the number of full sectors per slot.

DD Job Control Statement Processing

R
Is the remainder.

The minimum allowable buffer size in logical sectors is

N if R = 0.
N+1 if R divides evenly into LSS.
N+2 otherwise

To determine n, multiply the minimum allowable size in sectors by LSS.

TAPE

The buffer size is related to the record size (RCSZ) and record format (RCFM) specifications. The following guidelines apply for certain record formats:

- For variable-length blocked records
BKSZ must accommodate the largest block size in the file (including block and record descriptor words).
- For variable-length unblocked records
BKSZ must accommodate the largest record size (including block and record descriptor words).
- For fixed-length unblocked or blocked records
BKSZ must accommodate either record size or record size x blocking factor.

PRINTER

The buffer size determines the record size. For variable-length or undefined records, BKSZ is the length of the longest record (including block and record descriptor words for variable records). For programs using embedded control characters, BKSZ is n+1, where n is the length of the longest record to be printed.

CARD

The buffer size determines the record size. For variable-length unblocked records, BKSZ is the length of the longest record (including block and record descriptor words). When using binary mode, BKSZ is double the maximum record size.

C.2.3. Record Size (RCSZ)

This keyword parameter specifies the record size.

The following is device-dependent information regarding record size:

DISK/FORMAT LABEL DISKETTE/DATA SET LABEL DISKETTE

RCSZ specifies the length of each record in bytes. Since variable-length records are written in fixed size slots, RCSZ is the slot size (including the record descriptor word).

TAPE

RCSZ specifies the length of each record in bytes and is valid only in fixed-length record format.

C.2.4. Key Length (KLEn)

This keyword parameter specifies key lengths for an indexed file.

KLEn=L

Specifies the length of up to five keys, when n is from 1 to 5, inclusive, and L is from 1 to 80 bytes.

C.2.5. Key Location (KLOCn)

This keyword parameter specifies key locations for an indexed file.

KLOCn=L

Specifies the location of up to five keys, when n is from 1 to 5, inclusive, and L is the number of bytes (including the record descriptor word for variable length records) in the record preceding the key.

C.2.6. Index Buffer Size (INDS)

This keyword parameter specifies the index I/O buffer size.

INDS=n

Specifies the buffer size, where n is the number of bytes up to a maximum of 32,512 bytes and must be a multiple of 256.

C.2.7. Initial Space Allocation Percentages (SIZE, SIZE1, SIZE2)

CDM uses the file's record size and key size to estimate the ratio of data partition to index partition requirements. CDM uses this ratio to determine the percentages for initial space allocation of the data and index partitions. The percentages minimize the number of extent table entries that are required. This reduces the possibility of running out of extent table entries, resulting in fewer DM45 errors.

Prior to this release, you specified `SIZE=AUTO` on the file's DD JCL statement to allocate the percentages; now, allocation is performed automatically. If you specify `SIZE=AUTO`, it is ignored. The index partition is guaranteed to receive part of the initial allocation. Except for very small allocations (one, two, or three cylinders), part of the initial allocation is not assigned and is available for logical extensions.

The accuracy of the calculated percentages depends on the size of the file, the effects of roundoff, and how the file is loaded. For more efficient use of space in the index partition, load records in ascending key sequence, rather than in unordered key sequence. If the calculated percentages exhaust the extent table or result in wasted disk space, use the `SIZE1` and `SIZE2` parameters to specify explicit percentages for CDM to use.

`SIZE1=n SIZE2=m`

Specify both parameters: `n` represents the percentage to be used for the data partition and `m` represents the percentage to be used for the index partition. Their total must not exceed 100.

These specifications are effective only at file creation time; they are ignored if specified at other times. You create a file by

- Loading a newly allocated file

or

- Reloading (recreating) a file by using the `INIT` job control specification

If you used `SIZE` parameters to create a file, respecify them when you reload a file by using the `INIT` specification.

C.2.8. File Sharing Characteristics (ACCESS)

This keyword parameter specifies the disk file sharing characteristics. The characteristics indicate whether or not multiple logical access paths (LAPs) can access the physical file at the same time and the type of processing (read and/or write use) that each LAP can perform. See 5.4 for detailed information on disk file sharing and the `ACCESS` parameter.

C.2.9. Variable Sector Support (VSEC)

Both the data and index partitions specify a fixed sector size of 256 bytes. This is required for the sectorized devices (8416, 8417, 8418, 8419, and 8470 disk subsystems) which are formatted for a 256-byte sector. The selector channel devices (8430 and 8433 disk subsystems) have no such constraint. However, they are preformatted when the file is opened to accept the 256-byte sector size. This sectorization requires hardware overhead and thus decreases the effective capacity of the disk.

Variable sector support eliminates the problem. It allows you to create a disk file with a data partition physical sector size (PSS) that is larger than 256 bytes on the selector channel devices. Because the hardware overhead remains constant, the use of the larger PSS increases the effective capacity of the disk.

This keyword parameter is only valid at file creation time. It is not required for selector channel devices and is ignored if specified at other times. If it is specified for a sectorized device, it is ignored.

VSEC=n

Specifies the PSS where n is the size in bytes. To obtain the maximum benefit, n *should always* be a multiple of slot size.

VSEC=YES

Specifies that the PSS is to be automatically computed when the file is opened. The computed PSS will be the largest multiple of slot size.

Slot size is determined as follows:

- For a file with fixed-length records that contain an RCB, the slot size equals record size + 1.
- For other files, the slot size equals record size.

C.2.10. File Recovery Support (RECV)

When you perform operations such as adding or deleting records from a keyed file or updating records with a key change, the physical file structure changes constantly during program execution. If your program completes its run and the file is successfully closed, the file limits information contained in the file labels is updated.

If a system failure occurs, the file is not closed and the file limits information is not updated. A nonindexed file reverts to the state it was in before the file was opened, and any records added during the run are lost. The effect on an indexed file is more serious; the file's index may become compromised, and it is impossible to determine whether or not this has happened. DM24 or DM39 errors or premature end of file are indications that the index is compromised, but they may not appear immediately.

File recovery support eliminates these problems by writing the current file limits information to disk (the first sector of the data partition, PCA1) after each function that changes the limits information, such as record addition, keyed deletion, or update with key change. You should request recovery for your files, especially indexed files, because it saves you the inconvenience of rebuilding files and getting unexpected program errors. There is only a small overhead cost incurred on the functions that change file limits information.

The recovery facility is activated when the file is successfully closed, after file creation. If a system failure occurs during file creation, the file will have to be reloaded, because recovery was never activated. This ensures that no overhead is incurred during file creation. There is no overhead on read, update without key change, or nonindexed delete functions, regardless of when they occur.

When recovery is active for a file, and a system stop occurs

- A nonindexed file is recovered automatically. The file limits information reflects all of the records that were in the file at the time of the stop, and no records are lost.
- If the file is indexed, and CDM is in the middle of an index manipulation when the stop occurs, the index is compromised. The following error message appears the next time a program attempts to open the file:

```
DM66 FILE INACCESSIBLE TYPE=01
```

- If you do not receive this error on the first attempt to open the file after the stop, the file is totally recovered, the index is not compromised, and no records are lost. If you receive this error, use the RECV=FCE specification to open the file strictly to recreate it.
- Note that a VTOC print (SU VTP) reflects the correct number of records. The stop prevents label update, but correct information is maintained in the recovery block. SU VTP extracts information from the labels and the recovery block.

Use either the DMRECV SUPGEN parameter or the RECV DD JCL parameter to request recovery support. DMRECV lets you specify what kind of files will be created with recovery.

- DMRECV=YES - create all data files (except temporary files prefixed \$SCR or \$JOB, or system files prefixed with \$Y\$).
- DMRECV=INDEX - create only indexed data files (except temporary files prefixed \$SCR or \$JOB, or system files prefixed with \$Y\$).
- DMRECV=NO - create no files.

You can override these specifications on a file basis (except temporary files prefixed with \$SCR or \$JOB) by using the RECV parameter on the DD JCL statement. See the *Installation Guide* (UP-8839) for details on the DMRECV parameter.

RECV=YES

Creates the file with recovery support. It is not necessary to use this specification if the DMRECV parameter is set to request recovery for the file type in question.

This specification is effective only at file creation time; it is ignored if you specify it at other times. When you recreate a file by using the INIT specification, respecify this parameter if you used it to create the file the first time.

RECV=LOAD

Creates the file with recovery support and activates the recovery facility during file creation. Use this specification if you want to activate recovery when you load the file.

This specification is effective only at file creation time; it is ignored if you specify it at other times. When you reload the file using the INIT specification, you must specify this parameter again if you used it to create a file.

RECV=NO

Creates the file without recovery support. Use this specification when you use the SUPGEN parameter DMRECV=YES or DMRECV=INDEX and you do not want recovery for this file.

This specification is effective only at file creation time; it is ignored if you specify it at other times. When you reload the file by using the INIT specification, respecify this parameter if you used it to create a file.

RECV=FCE

Ignores the compromised file condition; you will not receive a DM66 TYPE=01 error. Use this specification only when you want to open a compromised file to read the data partition (and not the index) to recreate the file.

RECV=OFF

Deactivates the recovery facility during this job run. The recovery block will not be updated after each function that changes the file limits information. If the file is indexed and the file is not successfully closed (that is, system stop), you will receive a DM66 TYPE=01 error message the next time a program tries to open the file. For nonindexed files, added records will be lost.

This specification is effective only for ACCESS specifications of EXC or EXCR; it is not supported for ACCESS specifications of SADD or UCP. You can use this specification to avoid recovery overhead where, if the system stops, the file would be backed up anyway.

C.2.11. One Volume Online at a Time (VMNT)

This keyword parameter specifies whether or not a file is to be processed with only one volume online at a time.

VMNT=NO

Specifies that the file is to be processed with all volumes online. A file that is created in this manner must always be processed in this manner. Random operations are permitted.

VMNT=ONE

Specifies that the file is to be processed with only one volume online at a time. A file that is created in this manner must be processed in this manner. Nonkeyed random operations or keyed random output operations are not permitted.

C.2.12. Record Control Byte (RCB)

This keyword parameter specifies whether or not a record control byte (RCB) is present in each record.

RCB=YES

Each record contains an RCB. If the records are fixed length, the RCB is appended to the front of each record. If the records are variable length, the third byte in the 4-byte overhead is used as the RCB. The presence of RCBs allows you to logically delete records from the file by marking them as void records. The records are not physically removed from the file. The marking process consists of setting the high order bit in the RCB.

In addition, the presence of RCBs also provides the following:

- If you are retrieving records sequentially, the deleted records are skipped over.
- If you are retrieving records randomly and your search argument specifies the key or relative record number of a deleted record, it results in a no-find error condition.
- If you are outputting records randomly and you direct a record to a point beyond the last record in the file, any gap is filled with void records.
- If you are outputting records randomly and you direct a record to a point within the file limits, an error condition results if a valid (not deleted) record is at that point.

RCB=NO

Indicates that no record contains an RCB.

C.2.13. Offset Physical Sector (OFFSET)

Specify the OFFSET parameter when you want to process (convert) a data set label diskette created by an IBM (System/32 or System/34) utility. The first sector of such a diskette always contains file control information while the first sector of an OS/3 data set label diskette is always used for file data. OFFSET tells consolidated data management to position an IBM diskette to the second physical sector so that file processing can begin at the first logical record. For more information about converting IBM diskettes see the *System 32, 34 to OS/3 Conversion Guide* (UP-9318).

OFFSET=1

Specifies the first sector of a data set label diskette does not contain file data.

C.2.14. General Rewind Options (REWIND)

This keyword parameter specifies general tape rewinding options. If specified, OPRW (see C.2.15) and CLRW (see C.2.16) are ignored.

REWIND=UNLOAD

Specifies rewinding of the tape to the load point when the file is opened, and specifies unloading of the tape when the file is closed.

REWIND=NORWD

Specifies no tape rewinding when the file is opened, and tape positioning to the end of volume (between the two tape marks) when the file is closed. Note that open processing is identical to OPRW=NORWD, but close processing is *not* identical to CLRW=NORWD.

C.2.15. Rewinding at File Open (OPRW)

This keyword parameter specifies a rewinding option for file open.

OPRW=NORWD

Specifies no tape rewinding (to load point) before labels are checked during file open.

C.2.16. Rewinding at File Close (CLRW)

This keyword parameter specifies rewinding options for file close.

CLRW=NORWD

Specifies no tape rewinding after the file is closed.

CLRW=RWD

Specifies tape rewinding (to load point), but no unloading after the file is closed.

C.2.17. File Label Type (FILABL)

This keyword parameter specifies the label type (if any) for the file and is applicable to all volumes of a multivolume file.

FILABL=NO

Specifies that the file is unlabeled. This specification is not valid for ASCII files.

FILABL=STD

Specifies that the file has standard labels that conform to system conventions.

FILABL=NSTD

Specifies that the file has nonstandard labels. This specification is not valid for ASCII files.

C.2.18. Tape Marks (TPMARK)

This keyword parameter specifies a tape-marking option.

TPMARK=NO

Specifies that data management is not to write a tape mark preceding data on nonstandard labeled or unlabeled output files. Distinguishing between labels and data when reading nonstandard labeled files is your responsibility.

C.2.19. Restoring Initialized Files (RESTORE)

This keyword parameter enables you to restore a MIRAM disk file after it was unintentionally initialized.

RESTORE=YES

Beginning with OS/3 Release 12.0, data management will save the record count whenever a file is initialized.

RESTORE=YES allows you to restore a file to its original capacity. The number of records written to the file after initialization has overlaid the same number of original records. The original records that were overlaid are lost; however, the remainder of the original records are present and are accessible. It is recommended that you use RESTORE=YES if a file was unintentionally initialized and then copy that file to another file.

This specification will not work with a file that has been scratched and then reallocated into the same physical space it originally occupied or with a file that was unintentionally initialized in a release prior to OS/3 Release 12.0.

RESTORE=n

Specifies the number of records that data management is to read in order to restore the file. Use this specification to read the data partition of the file (PCA1) and copy or rebuild the file. The reasons for using this specification are

- If there is a need to restore a portion of the original file, n would represent the number of records to restore
- If the file was scratched and reallocated into the same physical space it originally occupied
- If the file was unintentionally initialized on a release prior to OS/3 Release 12.0

You must use a copy program, such as DATA or MILOAD, to copy the file or rebuild the index.

C.2.20. Disk Cache Support (CACHE)

This keyword parameter and the CACHE IOGEN parameter provide selective caching of MIRAM disk files. You may not wish to cache data from a file when:

- A file is accessed randomly. Caching may not improve performance, but may impact it.
- Performance is not a priority in file processing and you wish to reserve the cache buffer for other processing.

Use the CACHE IOGEN parameter to specify the files that are to be cached for the disk device.

- CACHE=YES - cache data from all files. YES is the default.
- CACHE=NO - do not cache data from any files.
- CACHE=NOMI - cache data from all but MIRAM data files.

For further information on the CACHE IOGEN parameter, refer to the *Installation Guide* (UP-8839).

Note that the REMOVE command can also be used to remove a device from the disk cache facility. For further information on the disk cache facility and descriptions of all cache commands, refer to the *Operations Guide* (UP-8859).

To override a YES or NOMI specification on a file basis, use the CACHE parameter in the DD JCL statement.

CACHE=NO

Specifies that the file be processed with no caching. Use DD CACHE=NO when the device is being cached (IOGEN CACHE=YES) and you do not want caching for this file.

CACHE=YES

Specifies that the file be processed with caching. Use DD CACHE=YES when you specify IOGEN CACHE=NOMI and you want caching for this file. This specification is unnecessary if you indicate IOGEN CACHE=YES. If you indicate IOGEN CACHE=NO, this specification is ignored. DD CACHE=YES applies only to MIRAM data files.

C.2.21. Suppressing Error Messages (MSGSUPP)

This keyword parameter allows you to suppress the display of the DM36 error message, the LB05 error message, or all error messages.

MSGSUPP=DM36

Specifies that the DM36 DUPLICATE RECORD error message is not to be displayed on the console or log.

MSGSUPP=LB05

Specifies that the LB05 MODULE NOT FOUND error message is not to be displayed on the console or log.

MSGSUPP=ALL

Specifies that all data management error messages are not to be displayed on the console or log.

Appendix D

Shared Code and Accelerated File Access

For each file type (disk, tape, card, and so on), there is a data management processing module invoked whenever a file of that type is accessed. These processing modules are shared, meaning that when a system or user program requests access of a particular file type, a copy of the corresponding module is loaded into main storage for use by any program requesting access of the same file type.

To improve the performance of consolidated data management when accessing files, you can make the processing modules for card, printer, tape, diskette, and disk files resident. A shared code module designated as resident is loaded when the operating system (actually, the supervisor) is loaded and is never moved or deleted. Consolidated data management recognizes that the module is resident and takes advantage of the fact that the module's location cannot change. Accelerated file access results because control can be transferred to a resident module faster than it can be to a nonresident module.

To designate a module as resident, you must specify the name of the module in the RESHARE parameter during generation of your supervisor (SUPGEN). The file types and corresponding module names you can specify are

<u>File Type</u>	<u>Module Name</u>
Disk	D3SM1110
Diskette	D3SM1110
Tape	DDST1110
Card	CDSIOJ00
Printer	PRSIOE00

Notes:

1. *Accelerated access is provided only for disk files with share requirements of EXC or SRDO. Files with share requirements of SRD, EXCR, or SADD, for example, are not provided with accelerated file access even if the disk module is resident. (See 5.4 for information about disk file share requirements.)*

2. *Acceleration is not provided for single-mount multivolume disk or diskette files or multivolume tape files.*

It isn't necessary to make all the modules resident; only those corresponding to the particular file type that you want accelerated access for. For more information about the RESHARE parameter, see the *Installation Guide* (UP-8839).

Appendix E

Data Management Debugging Facility

E.1. General Information

The data management debugging facility obtains documentation (a system dump) when an unexpected data management error occurs.

E.2. Console Command

The following console command will activate the debugging facility:

```
SE DE,DM,ees
```

where:

ee

Is the data management error code.

ss

Is the subcode received on the error message.

You must enter a 4-character code even if there is no subcode associated with the error code. For example, to activate the debug facility for a DM06 error, you enter

```
SE DE,DM,0600
```

For those error codes that have subcodes, you can enter 00 for the subcode if you want a system dump produced for any occurrence of the error code (independent of the subcode).

When the error condition is encountered, an automatic system dump is generated (with an error code of 3DE), and the job continues to execute. It does not require a debug supervisor, and it will not result in an HPR or abnormal job termination. It is automatically deactivated upon the first occurrence of the error condition.



Index

A

ACCESS parameter 5-21, 5-22, 5-24

ASCII

- card files 2-3
- code correspondence with EBCDIC, Hollerith B-1, B-3
- end-of-file/end-of-volume coincidence 4-12
- magnetic tape files 4-1
- record formats 4-16
- standard volume organization 4-11
- translation tables 2-4

ASCII/EBCDIC/Hollerith correspondence
B-1, B-3

ASCII standard volume organization
end-of-file/end-of-volume coincidence
4-14, 4-15

- multifile, multivolume 4-13
- multifile, single volume 4-12
- single file, multivolume 4-11
- single file, single volume 4-11

ASCII-to-EBCDIC translation table 2-4

Autoloader feature, 8420 diskette 5-19, 6-6

B

Block, data structure 1-4

C

Card file programming considerations 2-3

Card files

- input 2-2
- organization 2-1
- output 2-2
- processing 51- or 66-column cards 2-3
- programming considerations 2-3
- punch, functional characteristics A-2
- reader, functional characteristics A-1
- record formats 2-2, 2-3
- structure 2-1

Card input files 2-2

Card output files 2-2, 2-3

Card punch subsystem characteristics A-2

Card reader subsystem characteristics A-1

Coarse-level index, MIRAM disk files 5-8,
5-9

Code correspondences Appendix B

Column binary (image) code B-9

Compressed card code B-8

Consolidated data management

- definition 1-1
- program development cycle 1-3
- relationship to a program 1-2

D

Data set label diskette files

- file layout 6-3
- organization 6-1
- programming considerations 6-4
- record formats 6-2, 6-3
- recording mode 6-1

Data set label diskette file programming considerations

- creating a data set label diskette file 6-5, 6-6
- existing data set label diskette file 6-7
- volume preparation 6-7

Data structure

- block 1-4
- field 1-4
- file 1-4
- record 1-4
- volume 1-4

Data utilities, software component 1-10

DD job control statement

- description C-1
- parameters C-1

Debugging facility E-1

Device assignment set

- card files 2-3
- creating a disk file 5-16
- data set label diskette files 6-5, 6-6
- existing disk file 5-21
- extending an existing disk file 5-21
- format label diskette files 5-16
- magnetic tape files 4-19
- MIRAM disk files 5-16
- printer files 3-4
- remote disk file 5-22
- workstation files 7-2

Disk file organization

- indexed files 5-1
- MIRAM data partition 5-3
- MIRAM index partition entries 5-6

- MIRAM nonindexed file 5-3
- nonindexed files 5-1

Disk file programming considerations

- creating a disk file 5-16, 5-17
- existing disk file 5-18
- extending an existing disk file 5-21
- volume preparation 5-22

Disk file sharing 5-22

Disk files

- access methods 5-2
- data partition 5-4
- disk volume preparation 5-22
- estimating disk space requirements 5-10, 5-15
- file sharing 5-22
- index partition entries 5-8, 5-9
- index structure 5-9
- indexed 5-1
- MIRAM concepts 5-2
- MIRAM data record formats 5-6
- MIRAM file organization 5-4
- nonindexed 5-1
- organization 5-1
- programming considerations 5-16
- remote 5-21

Disk subsystem characteristics A-6

Disk volume preparation 5-22

Diskette files

- data set label 6-1
- diskette volume preparation 6-7
- file layout 6-3
- organization 6-1
- programming considerations 6-4
- record formats 6-2, 6-3
- recording mode 6-1, 6-2

Diskette subsystem characteristics A-6

DSKPRP routine 5-22, 6-7

DVC job control statement
card files 2-3
creating a data set label diskette file 6-4,
6-5
creating a disk file 5-16, 5-17
existing data set label diskette file 6-7
existing disk file 5-21
job control software component 1-9
logical unit number, tape devices 4-20
magnetic tape files 4-20
multivolume workstation file 7-2
optional devices 4-20
printer files 3-4
single-volume workstation file 7-2

E

EBCDIC

internal code, system 2-4
nonstandard volume organization 4-6
record formats 4-16
standard volume organization 4-2
unlabeled volume organization 4-9

EBCDIC/ASCII/Hollerith correspondence
B-1, B-3

EBCDIC-encoded magnetic tape files
end-of-file condition 4-2, 4-4
end-of-volume condition 4-2, 4-5
nonstandard labels 4-6, 4-8
standard labeled 4-2, 4-3, 4-4
standard volume organization 4-2
unlabeled 4-9

EBCDIC-to-ASCII translation table 2-4

End-of-file condition 4-2, 4-4, 4-15

End-of-volume condition 4-2, 4-5, 4-15

EOF1/EOF2 labels 4-14, 4-15

EOV1/EOV2 labels 4-14, 4-15

Error and exception handling 1-8

Error message format 1-8

Errors, return of control 1-8

EXT job control statement

creating data set label diskette file 6-6,
6-6
creating a disk file 5-16, 5-17
estimating disk space requirements 5-10,
5-15
existing data set label diskette file 6-7
existing disk file 5-18
extending an existing disk file 5-21
job control software component 1-9

F

Field, data structure 1-4

File, data structure 1-4

File organization

ASCII-encoded magnetic tape files 4-1
disk files 5-1
diskette files 6-1
EBCDIC-encoded magnetic tape files 4-1
magnetic tape files 4-1
MIRAM indexed files 5-3
nonindexed MIRAM files 5-3
printer files 3-1
punched card files 2-1
workstation files 7-1

File sharing

ACCESS parameter 5-24
DD job control statement C-1
disk files 5-22

Fine-level index, MIRAM disk files 5-8, 5-9,
5-8

Fixed-head area, 8417 disk 5-17

Format label diskette files 5-1

Functional characteristics, I/O devices
 card punch subsystems A-3
 card reader subsystems A-1
 disk subsystems A-7
 diskette subsystems A-7
 magnetic tape subsystems A-8
 printer subsystems A-4
 workstation subsystems A-9

H

Hollerith/ASCII/EBCDIC correspondence
 B-1, B-3

I

Image (column binary) code B-9

I/O devices, functional characteristics
 Appendix A

J

Job control software component 1-9

L

LBL job control statement
 checking volume/file serial numbers 4-22
 creating a data set label diskette file 6-4,
 6-5
 creating a disk file 5-16, 5-19
 creating volume/file serial numbers 4-22
 data management action to open file 4-23
 existing data set label diskette file 6-7
 existing disk file 5-21
 file creation date 4-24
 file expiration date 4-24
 file generation/version number 4-24
 file identifier 4-22
 file label information 4-21
 file sequence number 4-24
 job control software component 1-9
 volume/file serial numbers 4-21

LCB job control statement 3-5

LFD job control statement
 card files 2-3
 creating a data set label diskette file 6-5,
 6-6
 creating a disk file 5-16, 5-17
 existing data set label diskette file 6-7
 existing disk file 5-21
 extending an existing tape file 4-22, 4-25
 file sharing requirements 5-23
 job control software component 1-9
 logical file definition 4-22, 5-16, 6-5, 7-2
 magnetic tape files 4-25
 multivolume workstation file 7-2
 printer files 3-4
 single-volume workstation file 7-2
 tape device assignment 4-19

Logical access path (LAP) 5-23

M

Magnetic tape file programming
 considerations
 checking volume/file serial numbers 4-22
 creating multivolume files 4-27
 creating volume/file serial numbers 4-22
 extending tape files 4-25, 4-28
 file creation date 4-24
 file expiration date 4-24
 file generation/version numbers 4-24
 file identifier 4-22
 file label information 4-22
 file sequence number 4-24
 inhibiting volume serial number checking
 4-21
 logical file definition 4-25
 mode characteristics 4-27
 tape device assignment 4-20
 tape volume information 4-21
 tape volume preparation 4-26

Magnetic tape subsystem characteristics A-8

- Magnetic tape files
 - ASCII 4-1
 - EBCDIC 4-1
 - file organization 4-1
 - multifile volume 4-4, 4-5, 4-8, 4-12, 4-13, 4-15
 - multivolume file 4-11, 4-13
 - nonstandard EBCDIC 4-6
 - programming considerations 4-19
 - record formats 4-16
 - single file 4-3, 4-7, 4-9, 4-11
 - volume organization 4-1
 - Magnetic tape volume organization
 - ASCII end-of-file/end-of-volume coincidence 4-14, 4-15
 - ASCII standard volume 4-10, 4-11, 4-12, 4-13
 - EBCDIC nonstandard volume 4-6, 4-8
 - EBCDIC standard volume 4-2, 4-4, 4-5
 - EBCDIC unlabeled volume 4-9
 - Magnetic tape volume preparation
 - tape prep utility (TPREP) 4-26
 - VOL job control statement 4-26
 - Mid-level index, MIRAM disk files 5-8, 5-9
 - MIRAM characteristic files 5-3
 - MIRAM concepts 5-2
 - MIRAM disk file organization
 - data partition 5-4
 - estimating disk space 5-10, 5-15
 - index partition entries 5-8
 - index structure 5-9
 - MIRAM disk file job control considerations
 - creating a disk file 5-16
 - existing disk file 5-21
 - extending an existing disk file 5-21
 - volume preparation 5-22
 - MIRAM disk files
 - concepts 5-2
 - data partition 5-4
 - data record format 5-6
 - disk volume preparation 5-22
 - estimating disk space requirements 5-10, 5-15
 - file sharing 5-22
 - index partition entries 5-8
 - index structure 5-9
 - nonindexed 5-1
 - organization 5-4
 - programming considerations 5-16
- ## N
- Nonindexed files
 - disk 5-1
 - MIRAM disk 5-2, 5-3
- ## P
- Printer file job control considerations 3-4
 - Printer files
 - data on preprinted forms 3-2
 - load code buffer 3-4
 - organization 3-1
 - programming considerations 3-4
 - record formats 3-3
 - tabular data 3-1, 3-2
 - text 3-1
 - vertical format buffer 3-4
 - Printer subsystem characteristics A-4
 - Programming considerations
 - card files 2-3
 - data set label diskette files 6-4
 - disk files 5-16
 - magnetic tape files 4-19
 - MIRAM files 5-16
 - printer files 3-4
 - workstation files 7-1

R

- Record, data structure 1-4
- Record formats
 - card input files 2-2
 - card output files 2-2, 2-3
 - data set label diskette files 6-2, 6-3
 - magnetic tape files 4-16
 - MIRAM data records 5-6
 - printer files 3-3
 - spanning physical block or sector boundaries, MIRAM data records 5-5
 - workstation files 7-1
- Recording mode 6-1, 6-2
- Remote disk file 5-21

S

- Software, related
 - data utilities 1-10
 - job control 1-9
 - supervisor 1-10
 - system service programs (SSP) 1-8
- Supervisor, software component 1-9
- System service programs (SSP), software component 1-8

T

- Tape marks
 - EBCDIC nonstandard volume organization 4-6
 - EBCDIC standard volume organization 4-2
 - EBCDIC unlabeled volume organization 4-7

U

- User header labels
 - optional 4-6
 - standard 4-2
- User trailer labels
 - optional 4-6
 - standard 4-2

V

- VFB job control statement 3-4
- VOL job control statement
 - bypass volume serial number checking 4-21
 - data management action to open tape file 4-23
 - creating a data set label diskette file 6-5, 6-6
 - creating a disk file 5-16
 - creating multivolume files 4-27
 - existing data set label diskette file 6-6
 - existing disk file 5-18
 - job control software component 1-9
 - mode characteristics, tape volume 4-27
 - tape volume preparation 4-26
 - volume serial number specification 4-22

- Volume, data structure 1-4

W

- Workstation file job control considerations
 - multivolume workstation file 7-2
 - single-volume workstation file 7-2
- Workstation files
 - device assignment sets 7-2
 - organization 7-1
 - programming considerations 7-1
 - record formats 7-1
- Workstation subsystem characteristics A-8

UNISYS

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

(Document Title)

(Document No.)

(Revision No.)

(Update Level)

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage is necessary if mailed in the U.S.A.)
Thank you for your cooperation



FOLD



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

Unisys Corporation
E/MSG Product Information Development
PO Box 500 — E5-114
Blue Bell, PA 19422-9990



FOLD

NOTES



NOTES





