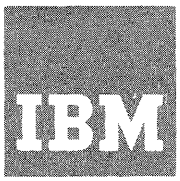
File Number S360-30
Order Number GC26-3746-1



Systems Reference Library

OS Data Management Services Guide

Release 21

This book describes the services provided by the operating system to allow the programmer to organize data into data sets on auxiliary storage devices, to read information from these data sets into main storage, and, after processing the information, to record it on auxiliary storage devices.

This book is intended for application programmers who write assembler–language programs that create and process data sets. It describes the assembler–language macro instructions used to request input and output operations. The format of the macro instructions is explained in *OS Data Management Macro Instructions,* GC26–3794, which should be used with this book.

In addition to describing the characteristics of data sets and direct–access storage devices, the book describes the techniques you can use to process sequential, partitioned, indexed sequential, and direct data sets.

This book assumes you have a basic knowledge of the operating system and of assembler language. Two books that contain information about these subjects are *OS Introduction,* GC28–6534, and *OS Assembler Language,* GC28–6514. It also assumes you are familiar with job control language, especially the DD statement, as described in *OS Job Control Language Reference,* GC28–6704.

This book does not discuss macro instructions used for the time sharing option or for graphics, teleprocessing, optical character readers, optical reader–sorters, or magnetic character readers. These macro instructions are discussed in separate publications that are listed in the *IBM System/360 and System/370 Bibliography,* GA22–6822.

Second Edition (February 1972)

This publication corresponds to release 21 of the operating system. It is a major revision of GC26–3746–0 and technical newsletters GN26–0624 and GN26–0631, which are now obsolete. A change is indicated by a vertical line in the margin to the left of the change. The major changes are listed in "Summary of Changes."

Information in this book changes from time to time. Before using this publication with IBM systems, consult the latest *IBM System/360 and 370 SRL Newsletter,* GN20–0360, for the editions that are current and applicable.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM Branch Office serving your locality.

Forms are provided at the back of this publication for reader's comments. If the forms have been removed, comments may be addressed to IBM Corporation, Programming Publications, Department D78, San Jose, California 95114. All comments become the property of IBM.

HOW TO USE THIS BOOK

If you know how to write assembler–language programs and use job control statements, you can use this book to write programs that create and process data sets. To use this book you must have basic knowledge of the operating system as contained in *OS Introduction,* GC28–6534, of assembler language as described in *OS Assembler Language,* GC28–6514, and of job control language (JCL) as explained in *OS Job Control Language Reference,* GC28–6704.

"Part 1: Introduction to Data Management" introduces you to the characteristics of data sets, how you name them, how the system catalogs them, and how you format the records in them. The format of tracks on a direct–access storage device is explained briefly. If you want to know more on this subject you can refer to *Introduction to IBM System/360 Direct–Access Storage Devices and Organization Methods,* GC20–1649.

Part 1 also describes the data control block (DCB) and the information it supplies to the operating system. Special processing routines that you specify in the DCB macro instruction are also explained in this section.

In "Part 2: Data Management Processing Procedures" there is an explanation of data–processing techniques that includes the macro instructions for the queued access technique and the basic access technique and the macro instructions for analyzing input and output errors. The section on data–processing techniques also tells how to select an access method and how to begin and end processing of a data set.

The section "Buffer Acquisition and Control" in Part 2 explains three different methods you can use to obtain buffers and the macro instructions you use with each method. This section also describes ways to control buffers: simple buffering and exchange buffering for the queued access technique, direct buffering and dynamic buffering for the basic access technique. In addition, for the queued access technique, there is an explanation of the four modes of moving the records in main storage: move mode, data mode, locate mode, and substitute mode. Macro instructions for controlling buffers are described here, too.

The next four sections of Part 2 concern processing data sets of four different types: a sequential data set, a partitioned data set, an indexed sequential data set, and a direct data set. They explain the organization of the data sets and the macro instructions used to process them. In the examples the macro instructions are coded in just enough detail to make the examples clear. For a complete description of the operands and options available, see *OS Data Management Macro Instructions,* GC26–3794.

"Part 3: Data Set Disposition and Space Allocation" tells you how to figure the amount of space you need for a data set on a direct–access storage device and how to request that space in your JCL DD statement. You are given special directions for allocating space for a partitioned data set and an indexed sequential data set. Part 3 also tells how to indicate in the JCL DD statement the status of the data set at the beginning of and during processing and how to indicate what you want the system to do with the data set when processing has terminated. You also are told how to use the DD statement to route the data set to a system output writer, to concatenate data sets, to catalog data sets, and to protect confidential data sets.

Appendix A describes data set labeling. Appendix B explains control characters you can use to control card punches and printers. Appendix C explains special

considerations for writing programs for the 3505 Card Reader and the 3525 Card Punch.

The following books are referred to in the text:

OS Data Management for System Programmers, GC28–6550, is for system programmers who are responsible for maintaining, updating, and extending the data management libraries of the operating system.

OS Messages and Codes, GC28–6631, explains the system completion codes and error messages issued by the operating system.

OS MFT Guide, GC27–6939, and *OS MVT Guide,* GC28–6720, provide introductory material for programmers not familiar with the multiprogramming with a fixed number of tasks (MFT) and multiprogramming with a variable number of tasks (MVT) versions of the operating system.

OS Service Aids, GC28–6719, aids system programmers and persons maintaining OS in diagnosis of system or application program failures.

OS Sort/Merge, GC28–6543, explains how to use the sort/merge program.

OS Supervisor Services and Macro Instructions, GC28–6646, explains how to reserve system resources for the exclusive control of your program and how to restart your program from a checkpoint if the system fails.

OS System Control Blocks, GC28–6628, shows the format of the control blocks used by more than one component of the operating system.

OS System Generation, GC28–6554, describes the process you use to create or modify an operating system.

OS Tape Labels, GC28–6680, explains how the system processes tapes with IBM standard labels, American National Standard labels, nonstandard labels, or no labels.

OS Utilities, GC28–6586, describes how to use operating system utility programs that are used by programmers responsible for organizing and maintaining operating system data.

CONTENTS

FIGURES

SUMMARY OF CHANGES FOR RELEASE 21

DCB ABEND Exit

A description of the DCB ABEND exit routine has been added to the section "Exits to Special Processing Routines" in Part 1.

DOS Tapes

Special considerations for using the CNTRL, BSP, and POINT macro instructions in processing DOS tapes under OS are in the section "Device Control." A restriction is listed in the section "Chained Scheduling for I/O Operations."

IEHPROGM Data Set Protection Feature

This new feature of the IEHPROGM utility program is described in the section "Controlling Confidential Data — Password Protection."

Time Sharing Option (TSO)

TSO information has been removed. See *Time Sharing Option Guide to Writing a Terminal Monitor Program or a Command Processor,* GC28–6764.

3400–Series Magnetic–Tape Units

Recording densities are given in the section "Data Format — Device Type Considerations."

3505 Card Reader and 3525 Card Punch

Special programming considerations for these devices are in Appendix C. Restrictions appear in the sections "Opening and Closing a Data Set," "Data Format — Device Type Considerations," and "Chained Scheduling for I/O Operations."

ACRONYMS USED IN THIS BOOK

A	ANSI control code (value of RECFM)
ABE	**ab**normal **e**nd (value of EROPT)
ABEND	**ab**normal **end** (macro instruction)
ACC	**acc**ept erroneous block (value of EROPT)
AFF	**aff**inity (channel separation parameter of DD statement or unit affinity value of UNIT)
ANSI	American National Standards Institute
ASCII	American National Standard Code for Information Interchange
ABSTR	**abs**olute **tr**ack (value of SPACE)
AUL	American National Standard or User labels (value of LABEL)
B	**b**locked records (value of RECFM)
BCDIC	**B**inary **C**oded **D**ecimal **I**nterchange **C**ode
BDAM	**b**asic **d**irect **a**ccess **m**ethod
BDW	**b**lock **d**escriptor **w**ord
BFALN	**buf**fer **al**ign**n**ment (operand of DCB)
BFTEK	**buf**fer **te**chnique (operand of DCB)
BISAM	**b**asic **i**ndexed **s**equential **a**ccess **m**ethod
BLDL	**bu**ild **l**ist (macro instruction)
BLKSIZE	**blk**size (operand of DCB)
BPAM	**b**asic **p**artitioned **a**ccess **m**ethod
BSAM	**b**asic **s**equential **a**ccess **m**ethod
BSM	**b**ack**s**pace past tape**m**ark and forward space over tapemark (operand of CNTRL)
BSP	**b**ack**sp**ace one block (macro instruction)
BSR	**b**ack**s**pace over a specified number of blocks (**r**ecords) (operand of CNTRL)
BUFCB	**buf**fer pool control **b**lock (operand of DCB)
BUFL	**buf**fer **l**ength (operand of DCB)
BUFNO	**buf**fer **n**umber (operand of DCB)
BUFOFF	**buf**fer **off**set (length of ASCII block prefix by which the buffer is offset; operand of DCB)
CCW	**c**hannel **c**ommand **w**ord
CONTIG	**contig**uous space allocation (value of SPACE)
CPU	**c**entral **p**rocessing **u**nit
CSW	**c**hannel **s**tatus **w**ord
CYLOFL	number of tracks for **cyl**inder **ov**er**fl**ow records (operand of DCB)
D	format–**D** (ASCII variable–length) records (value of RECFM)
DA	**d**irect–**a**ccess (value of DEVD or DSORG)
DAU	**d**irect–**a**ccess **u**nmovable data set (value of DSORG)
DCB	**d**ata **c**ontrol **b**lock (control block name or macro instruction)
DCBD	**d**ata **c**ontrol **b**lock **d**ummy section macro instruction
DD	**d**ata **d**efinition
DEB	**d**ata **e**xtent **b**lock
DECB	**d**ata **e**vent **c**ontrol **b**lock
DEN	magnetic tape **den**sity (operand of DCB)
DEVD	**dev**ice–**d**ependent (operand of DCB)
DISP	data set **disp**osition (parameter of DD statement)

DSCB	data set control block
DSORG	data set organization (operand of DCB)
EBCDIC	Extended Binary Coded Decimal Interchange Code
EODAD	end–of–data set exit routine address (operand of DCB)
EOF	end–of–file
EOV	end–of–volume
EROPT	error options (operand of DCB)
ESETL	end sequential retrieval (QISAM macro instruction)
EXCP	execute channel program (macro instruction)
EXLST	exit list (operand of DCB)
F	fixed–length records (value of RECFM)
FB	fixed–length, blocked records (value of RECFM)
FBS	fixed–length, blocked, standard records (value of RECFM)
FBT	fixed–length, blocked records with track overflow option (value of RECFM)
FCB	forms control buffer
FEOV	force end–of–volume (macro instruction)
FS	fixed–length, standard records (value of RECFM)
FSM	forward space past tapemark and backspace over tapemark (operand of CNTRL)
FSR	forward space over a specified number of blocks (records) (operand of CNTRL)
GL	GET macro, locate mode (value of MACRF)
GM	GET macro, move mode (value of MACRF)
HA	home address
I/O	input/output
INOUT	input then output (operand of OPEN)
IOB	input/output block
IPL	initial program load
IRG	interrecord gap
IS	indexed sequential (value of DSORG)
ISAM	indexed sequential access method
ISU	indexed sequential unmovable (value of DSORG)
JCL	job control language
JFCB	job file control block
KEYLEN	key length (operand of DCB)
LRECL	logical record length (operand of DCB)
M	machine control code (value of RECFM)
MACRF	macro instruction form (operand of DCB)
MFT	multiprogramming with a fixed number of tasks
MOD	modify data set (value of DISP)
MSHI	main storage for highest–level index (operand of DCB)
MSWA	main storage for work area (operand of DCB)
MVT	multiprogramming with a variable number of tasks
NCP	number of channel programs (operand of DCB)
NOPWREAD	no password to read a data set (value of LABEL)
NRZI	non–return–to–zero–inverse (tape recording mode)
NSL	nonstandard label (value of LABEL)

NTM	number of tracks in cylinder index for each entry in lowest level of master index (operand of DCB)
OMR	optical mark read on 3505 Card Reader
OPTCD	optional services code (operand of DCB)
OS	operating system
OUTIN	output then input (operand of OPEN)
PCI	program–controlled interruption
PDS	partitioned data set
PE	phase encoding (tape recording mode)
PL	PUT macro, locate mode (value of MACRF)
PM	PUT macro, move mode (value of MACRF)
PO	partitioned organization (value of DSORG)
POU	partitioned organization unmovable (value of DSORG)
PRTSP	printer line spacing (operand of DCB)
PS	physical sequential (value of DSORG)
PSU	physical sequential unmovable (value of DSORG)
QISAM	queued indexed sequential access methods
QSAM	queued sequential access method
RCE	read column eliminate for 3505 Card Reader and 3525 Card Punch
RDBACK	read backward (operand of OPEN)
RDW	record descriptor word
RECFM	record format (operand of DCB)
RKP	relative key position (operand of DCB)
RPS	rotational position sensing
S	standard format records (value of RECFM)
SDW	segment descriptor word
SEP	separation (channel separation parameter of DD statement or unit separation value of UNIT)
SER	volume serial number (value of VOLUME)
SETL	set lower limit of sequential retrieval (QISAM macro instruction)
SF	sequential forward (operand of READ or WRITE)
SK	skip to a printer channel (operand of CNTRL)
SKP	skip erroneous block (value of EROPT)
SL	IBM standard labels (value of LABEL)
SMSI	size of main–storage area for highest–level index (operand of DCB)
SMSW	size of main–storage work area (operand of DCB)
SP	space lines on a printer (operand of CNTRL)
SS	select stacker on card reader (operand of CNTRL)
SUL	IBM standard and user labels (value of LABEL)
SYNAD	synchronous error routine address (operand of DCB)
SYSIN	system input stream
SYSOUT	system output stream
T	track overflow option (value of RECFM)
TIOT	task I/O table
TRTCH	track recording technique (operand of DCB)
U	undefined length records (value of RECFM)
UCS	universal character set
UHL	user header label
UTL	user trailer label

V	format–**V** (variable–length) records (value of RECFM)
VB	variable–length, **b**locked records (value of RECFM)
VBS	variable–length, **b**locked, **s**panned records (value of RECFM)
VTOC	**v**olume **t**able **o**f **c**ontents

PART 1: INTRODUCTION TO DATA MANAGEMENT

Data Set Characteristics

The data management programs of the operating system (OS) help you achieve maximum efficiency in managing the mass of data associated with the many programs that are processed at your installation by providing systematic and effective means of organizing, identifying, storing, cataloging, and retrieving all data, including programs, processed by the operating system.

Data set storage control, along with an extensive catalog system, makes it possible for you to retrieve data by symbolic name alone, without specifying device types and volume serial numbers. In freeing computer personnel from maintaining involved volume serial number inventory lists of stored data and programs, the catalog reduces manual intervention and the likelihood of human error.

Data sets stored within the cataloging system can be classified according to installation needs. For example, a sales department could classify the data it uses by geographic area, by individual salesman, or by any other logical plan.

The cataloging system also makes it possible for you to classify successive generations or updates of related data. These generations can be given an identical name and subsequently be referred to relative to the current generation. The system automatically maintains a list of the most recent generations.

You can request data from a direct–access volume, a remote terminal, or a tape volume, and data organized sequentially or directly, in essentially the same way. In addition, data management provides:

- Allocation of space on direct–access volumes. Flexibility and efficiency of direct–access devices are improved through greater use of available space.
- Automatic retrieval of data sets by name alone.
- Freedom to defer specifications such as buffer length, blocksize, and device type until a job is submitted for processing. This permits the creation of programs that are in many ways independent of their operating environment.

Control of confidential data is provided by the data set security part of the operating system. You can prevent unauthorized access to payroll data, sales forecast data, and all other data sets that require special security attention. An individual can use a security–protected data set only after furnishing a predefined password.

Input/output routines are provided to efficiently schedule and control the transfer of data between main storage and input/output devices. Routines are available to:

- Read data
- Write data
- Translate data from ASCII (American National Standard Code for Information Interchange) to EBCDIC (Extended Binary Coded Decimal Interchange Code) and back
- Block and deblock records
- Overlap reading, writing, and processing operations

- Read and verify volume and data set labels
- Write data set labels
- Automatically position and reposition volumes
- Detect error conditions and correct them when possible
- Provide exits to user–written error and label routines

OS data management programs also provide for a variety of methods for gaining access to a data set. The methods are based on data set organization and data access technique.

OS data sets can be organized in four ways:

- *Sequential:* Records are placed in physical rather than logical sequence. Given one record, the location of the next record is determined by its physical position in the data set. Sequential organization is used for all magnetic–tape devices, and may be selected for direct–access devices. Punched tape, punched cards, and printed output are sequentially organized.
- *Indexed Sequential:* Records are arranged in sequence, according to a key that is a part of every record, on the tracks of a direct–access volume. An index or set of indexes maintained by the system gives the location of certain principal records. This permits direct as well as sequential access to any record.
- *Direct:* The records within the data set, which must be on a direct–access volume, may be organized in any manner you choose. All space allocated to the data set is available for data records. No space is required for indexes. You specify addresses by which records are stored and retrieved directly.
- *Partitioned:* Independent groups of sequentially organized records, called members, are in direct–access storage. Each member has a simple name stored in a directory that is part of the data set and contains the location of the member's starting point. Partitioned data sets are generally used to store programs. As a result, they are often referred to as libraries.

Requests for input/output operations on data sets through macro instructions employ two techniques: the technique for *queued access* and the technique for *basic access.* Each technique is identified according to its treatment of buffering and synchronization of input and output with processing. The combination of an access technique and a given data set organization is called an *access method.* In choosing an access method for a data set, therefore, you must consider not only its organization, but also what you need to specify through macro instructions. Also, you may choose a data organization according to the access techniques and processing capabilities available.

The code generated by the macro instructions for both techniques is optionally reenterable depending on the form in which parameters are expressed.

In addition to the access methods provided by the operating system, an elementary access technique called *execute channel program* is also provided. To use this technique, you must establish your own system for organizing, storing, and retrieving data. Its primary advantage is the complete flexibility it allows you in using the computer directly.

An important feature of data management is that much of the detailed information needed to store and retrieve data, such as device type, buffer processing technique, and format of output records need not be supplied until the job is ready to be executed.

This device independence permits changes to those specifications to be made without changes in the program. Therefore, you may design and test a program without knowing the exact input/output devices that will be used when it is executed.

Device independence is a feature of both access techniques for processing a sequential data set. To some extent, you determine the degree of device independence achieved. Many useful device–dependent features are available as part of certain macro instructions, and achieving device independence requires some selectivity in their use.

Data Set Identification

Any information that is a named, organized collection of logically related records can be classified as a data set. The information is not restricted to a specific type, purpose, or storage medium. A data set may be, for example, a source program, a library of macro instructions, or a file of data records used by a processing program.

Whenever you indicate that a new data set is to be created and placed on auxiliary storage, you (or the operating system) must give the data set a name. The data set name identifies a group of records as a data set. All data sets recognized by name (referred to without volume identification) and all data sets residing on a given volume must be distinguished from one another by unique names. To assist in this, the system provides a means of qualifying data set names.

A data set name is one simple name or a series of simple names joined together so that each represents a level of qualification. For example, the data set name DEPT58.SMITH.DATA3 is composed of three simple names. Proceeding from the left, each simple name is a category within which the next simple name is a subcategory.

Each simple name consists of from 1 to 8 alphameric characters, the first of which must be alphabetic. The special character period (.) separates simple names from each other. Including all simple names and periods, the length of the data set name must not exceed 44 characters. Thus, a maximum of 22 simple names can make up a data set name.

To permit different executions of a program to process different data sets without program reassembly, the data set is not referred to by name in the processing program. When the program is executed, the data set name and other pertinent information (such as unit type and volume serial number) are specified in a job control statement called the *data definition (DD)* statement. To gain access to the data set during processing, reference is made to a *data control block (DCB)* associated with the name of the DD statement. Space for a data control block, which specifies the particular data set to be used, is reserved by a DCB macro instruction when your program is assembled.

Data Set Storage

System/360 and System/370 provide a variety of devices for collecting, storing, and distributing data. Despite the variety, the devices have many common characteristics. The generic term *volume* is used to refer to a standard unit of auxiliary storage. A volume may be any one of the following:

- A reel of magnetic tape
- A disk pack
- A bin in a data cell

- A drum
- That part of an IBM 2302 disk storage device served by one access mechanism (the device has either two or four volumes in all)

Each data set stored on a volume has its name, location, organization, and other control information stored in the *data set label* or *volume table of contents* (for direct–access volumes only). Thus, when the name of the data set and the volume on which it is stored are made known to the operating system, a complete description of the data set, including its location on the volume, can be retrieved. Then, the data itself can be retrieved, or new data added to the data set.

Various groups of labels are used to identify magnetic–tape and direct–access volumes, as well as the data sets they contain. Magnetic–tape volumes can have standard or nonstandard labels, or they can be unlabeled. Direct–access volumes must use standard labels. Standard labels include a volume label, a data set label for each data set, and optional user labels.

Keeping track of the volume on which a particular data set resides can be a burden and a source of error. To alleviate this problem, the system provides for automatic cataloging of data sets. The system can retrieve a cataloged data set if given only the name of the data set. If the name is qualified, each qualifier corresponds to one of the indexes in the catalog. For example, the system finds the data set DEPT58.SMITH.DATA3 by searching a master index to determine the location of the index name DEPT58, by searching that index to find the location of the index SMITH, and by searching that index for DATA3 to find the identification of the volume containing the data set.

By use of the catalog, collections of data sets related by a common external name and the time sequence in which they were cataloged (their generation) can be identified; they are called *generation data groups*. For example, a data set name LAB.PAYROLL(0) refers to the most recent data set of the group; LAB.PAYROLL(–1) refers to the second most recent data set, etc. The same data set names can be used repeatedly with no requirement to keep track of the volume serial numbers used.

Direct–Access Volumes

Direct–access volumes are used to store executable programs, including the operating system itself. Direct–access storage is also used for data and for temporary working storage. One direct–access storage volume may be used for many different data sets, and space on it may be reallocated and reused. A volume table of contents (VTOC) is used to account for each data set and available space on the volume.

Each direct–access volume is identified by a volume label, which is usually stored in track 0 of cylinder 0. You may specify up to seven additional labels, located after the standard volume label, for further identification.

The VTOC is a data set consisting of *data set control blocks (DSCBs)* that describe the contents of the direct–access volume. The VTOC can contain seven kinds of DSCBs, each with a different purpose and a different format number. *OS System Control Blocks* describes the seven kinds of DSCBs, their purposes, and their formats.

Each direct–access volume is initialized by a utility program before being used on the system. The initialization program generates the volume label and constructs the table of contents. For additional information on direct–access labels, see "Appendix A: Direct–Access Labels."

When a data set is to be stored on a direct–access volume, you must supply the operating system with the amount of space to be allocated to the data set, expressed in blocks, tracks, or cylinders. Space allocation can be independent of device type if the request is expressed in blocks. If the request is made in tracks or cylinders, you must be aware of such device considerations as cylinder capacity and track size.

Magnetic–Tape Volumes

Because of the sequential organization of magnetic–tape devices, the operating system does not require space allocation procedures comparable to those for direct–access devices. When a new data set is to be placed on a magnetic–tape volume, you must specify the data set sequence number if it is not the first data set on the reel. OS positions a volume with IBM standard labels, American National Standard labels, or no labels so that the data set can be read or written. If the data set has nonstandard labels, you must provide for volume positioning in your nonstandard–label–processing routines. All data sets stored on a given magnetic–tape volume must be recorded in the same density.

When a data set is to be stored on an unlabeled tape volume and you have not specified a volume serial number, the system assigns a serial number to that volume and to any additional volumes required for the data set. Each such volume is assigned a serial number of the form Lxxxyy where xxx indicates the data set sequence number from IPL to IPL and yy indicates the volume sequence number for the data set. If you specify volume serial numbers for unlabeled volumes on which a data set is to be stored, the system assigns volume serial numbers to any additional volumes required. If data sets residing on unlabeled volumes are to be cataloged or passed, you should specify the volume serial numbers for the volumes required. This will prevent data sets residing on different volumes from being cataloged or passed under identical volume serial numbers. Retrieval of such data sets could result in unpredictable errors.

Each data set and each data set label group on magnetic tape that is to be processed by the operating system must be followed by a tapemark. Tapemarks cannot exist within a data set. When the operating system is used to create a tape with standard labels or no labels, all tapemarks are automatically written. Two tapemarks are written after the last trailer label group on a volume to indicate the last data set on the volume. On an unlabeled volume, the two tapemarks are written after the last data set.

When the operating system is used to create a tape data set with nonstandard labels, the delimiting tapemarks are not written. If the data set is to be retrieved by the operating system, those tapemarks must be written by your nonstandard–label–processing routine. Otherwise, tapemarks are not required after nonstandard labels since positioning of the tape volumes must be handled by installation routines.

For more information on labels for magnetic–tape volumes, refer to *OS Tape Labels.*

The data on magnetic–tape volumes can be in either EBCDIC or ASCII. ASCII is a 7–bit code consisting of 128 characters. It permits data on magnetic tape to be transferred from one computer to another even though the two computers may be products of different manufacturers.

Data management support of ASCII and of American National Standard tape labels is such that data management can translate records on input tapes in ASCII into EBCDIC for internal processing and translate the EBCDIC back into ASCII for output. Records on such input tapes may be sorted into ASCII collating sequence, as explained in *OS Sort/Merge.*

Data Set Record Formats

A data set is composed of a collection of records that normally have some logical relation to one another. The record is the basic unit of information used by a processing program. It might be a single character, all information resulting from a given business transaction, or measurements recorded at a given point in an experiment. Much data processing consists of reading, processing, and writing individual records.

The process of grouping a number of records before writing them on a volume is referred to as *blocking*. A *block* is made up of the data between interrecord gaps (IRGs). Each block can consist of one or more records. Blocking conserves storage space on the volume because it reduces the number of IRGs in the data set. In many cases, blocking also increases processing efficiency by reducing the number of input/output operations required to process a data set.

Records may be in one of four formats: fixed–length (format–F), variable–length for data in EBCDIC (format–V), variable–length for data to be translated to or from ASCII (format–D), or undefined–length (format–U). The main consideration in the selection of a record format is the nature of the data set itself. You must know the type of input your program will receive and the type of output it will produce. Selection of a record format is based on this knowledge, as well as on an understanding of the input/output devices that are used to contain the data set and the access method used to read and write the data records. The record format of a data set is indicated in the data control block according to specifications in the DCB macro instruction, the DD statement, or the data set label.

For ASCII tapes, data can be in format–F, format–D, and format–U with the restrictions noted under "Fixed–Length Records, ASCII tapes," "Variable–Length Records—Format D," and "Undefined–Length Records." When data management reads records from ASCII tapes, it translates the records into EBCDIC. When data management writes records onto ASCII tapes, it translates the records into ASCII. Because you use input records after they are translated and because output records are translated when you ask data management to write them, you work only with EBCDIC.

Note: There is no minimum requirement for blocksize; however, if a data check occurs on a magnetic–tape device, any record shorter than 12 bytes in a Read operation or 18 bytes in a Write operation is treated as a noise record and lost. No check for noise is made unless a data check occurs. The sort/merge program does not accept records shorter than 18 bytes.

Fixed–Length Records

The size of fixed–length (format–F) records, shown in Figure 1, is constant for all records in the data set. The number of records within a block is constant for every block in the data set, unless the data set contains truncated (short) blocks. If the data set contains unblocked format–F records, one record constitutes one block.

The system automatically performs physical length checking on blocked format–F records, making allowance for truncated blocks. Because the channel and interruption system can be used to accommodate length checking, and the blocking and deblocking are based on a constant record length, format–F records can be processed faster than format–V records.

Format–F records are shown in Figure 1. The optional control character (C), used for stacker selection or carriage control, may be included in each record to be printed or punched.

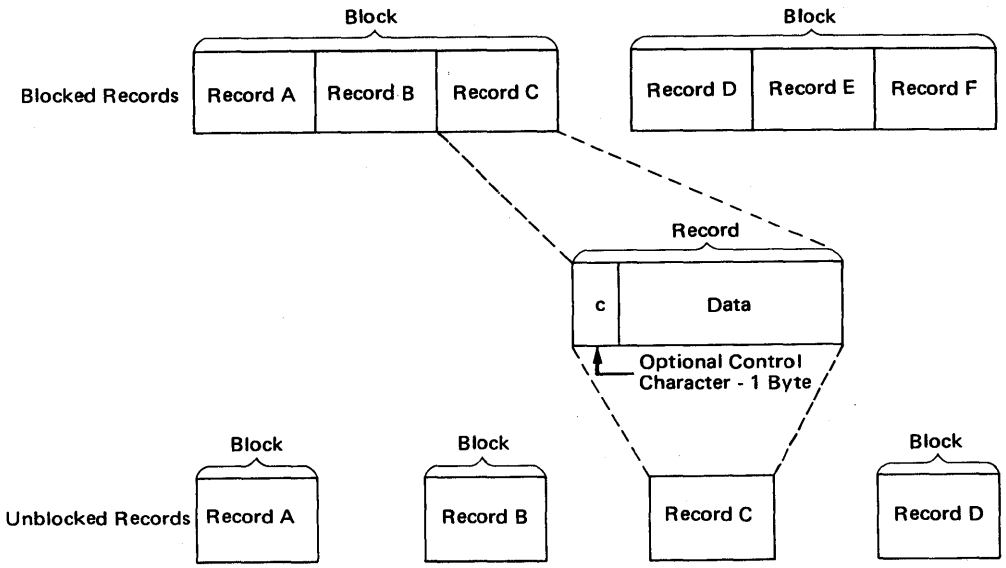


Figure 1. Fixed–Length Records

Fixed–Length Records, Standard Format: During creation of a sequential data set (to be processed by BSAM or QSAM) with fixed–length records, the RECFM subparameter of the DCB macro instruction may specify a standard format (RECFM=FS or FBS). A standard–format data set must conform to the following specifications:

- All records in the data set are format–F records.
- No block except the last block is truncated. (With BSAM you must ensure that this specification is met.)
- Every track except the last one contains the same number of blocks.
- Every track except the last one is filled to capacity as determined by the track capacity formula established for the device. (These formulas are presented in Part 3 of this book under "Allocating Space on Direct–Access Volumes.")

A sequential data set with fixed–length records having a standard format can be read more efficiently than a data set that doesn't specify a standard format. This efficiency is possible because the system is able to determine the address of each record to be read because each track contains the same number of blocks.

You should never extend a data set of this type (by coding DISP=MOD) if the last block is truncated, because the extension will cause the data set to contain a truncated block that isn't the last block. This type of data set on magnetic tape should not be read backward, because then the data set would begin with a truncated block. Consequently, you will probably not want to use this type of data set with magnetic tape. If you use one of the basic access techniques with this type of data set, you should not specify that the track overflow feature is to be used with the data set.

If at any time the characteristics of your data set are altered from the specifications described above, then the data set should no longer be processed with the standard format specification.

Fixed–Length Records, ASCII Tapes: For ASCII tapes, format–F records are the same as described above, with two exceptions:

- Control characters, if present, must be American National Standards Institute (ANSI) control characters.
- Records or blocks of records can contain block prefixes.

Figure 2 shows the format of fixed–length records for ASCII tapes and where control characters and block prefixes go if they exist.

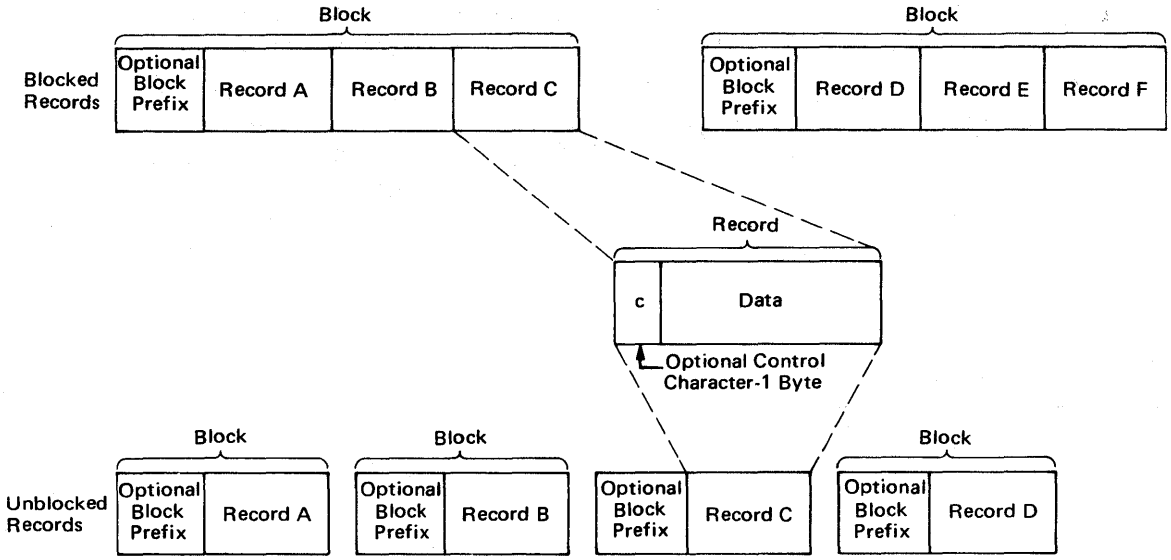


Figure 2. Fixed–Length Records for ASCII Tapes

The block prefix can vary in length from 0 to 99 bytes but the length must be constant for the data set being processed. For blocked records, the block prefix precedes the first logical record. For unblocked records, the block prefix precedes each logical record.

Using QSAM and BSAM to read records with block prefixes requires that you specify the BUFOFF operand in the DCB. When using QSAM, you cannot read the block prefix on input. When using BSAM, you must account for the block prefix on both input and output. When using either QSAM or BSAM, you must account for the length of the block prefix in the BLKSIZE and BUFL operands of the DCB.

When you use BSAM on output records, the operating system does not recognize a block prefix. Therefore, if you want a block prefix, it must be part of your record. Note that you cannot include block prefixes in QSAM output records.

The block prefix can contain any data you want, but you must avoid using data types such as binary, packed decimal, and floating–point that cannot be translated into ASCII.

For more information about control characters, refer to "Control Character" and to "Appendix B: Control Characters."

Variable–Length Records

The variable–length record formats are format V and format D. Format–V records can be spanned; that is, records can be larger than the blocksize, as described below. Format–D records are used for ASCII tape data sets and cannot be spanned.

Variable–Length Records — Format V: Format V provides for variable–length records, variable–length record segments, each of which describes its own characteristics, and variable–length blocks of such records or record segments. Except when variable–length track overflow records are specified for rotational position sensing, the control program performs length checking of the block and uses the record or segment length information in blocking and deblocking. The first 4 bytes of each record, record segment, or block make up a descriptor word containing control information. You must allow for these additional 4 bytes in both your input and output buffers.

Block Descriptor Word: A variable–length block consists of a block descriptor word (BDW) followed by one or more logical records or record segments. The block descriptor word is a 4–byte field that describes the block. The first 2 bytes specify the block length ('LL') — 4 bytes for the BDW plus the total length of all records or segments within the block. This length can be from 8 to 32,760 bytes or, when you are using WRITE with tape, from 18 to 32,760. The third and fourth bytes are reserved for future system use and must be 0. If the system does your blocking — that is, when you use the queued access technique — the operating system automatically provides the BDW when it writes the data set. If you do your own blocking — that is, when you use the basic access technique — you must supply the BDW.

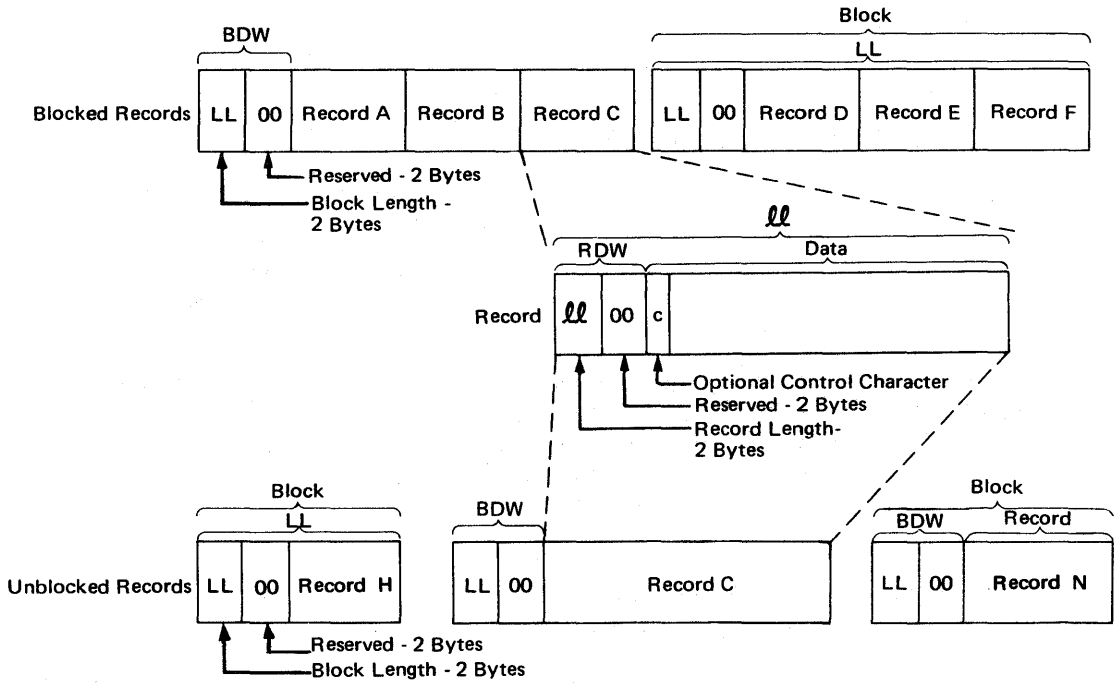


Figure 3. Variable–Length Records

Record Descriptor Word: A variable–length logical record consists of a record descriptor word (RDW) followed by the data. The record descriptor word is a 4–byte field describing the record. The first 2 bytes contain the length ('ll') of the logical record (including the 4–byte RDW). The length can be from 4 to 32,756. For information about processing a sequential data set, see "Data Format — Device Type Considerations." All bits of the third and fourth bytes must be 0, as other values are used for spanned records. For output, you must provide the RDW except in data mode for spanned records (described under "Buffer Control"). For output in data mode, you must provide the total data length in the physical record length field (DCBPRECL) of the DCB. For input, the operating system provides the RDW except in data mode. In data mode, the system passes the record length to your program in the logical record length field (DCBLRECL) of the DCB. The optional control character (C) may be specified as the fifth byte of each record. The RDW and the control character, if specified, are not punched or printed.

Figure 3 shows blocked and unblocked variable–length records without spanning.

Spanned Variable–Length Records (Sequential Access Methods): The spanning feature of the queued and basic sequential access methods enables you to create and process variable–length logical records that are larger than one physical block and/or to pack blocks with variable–length records by splitting the records into segments so that they can be written into more than one block, as shown in Figure 4.

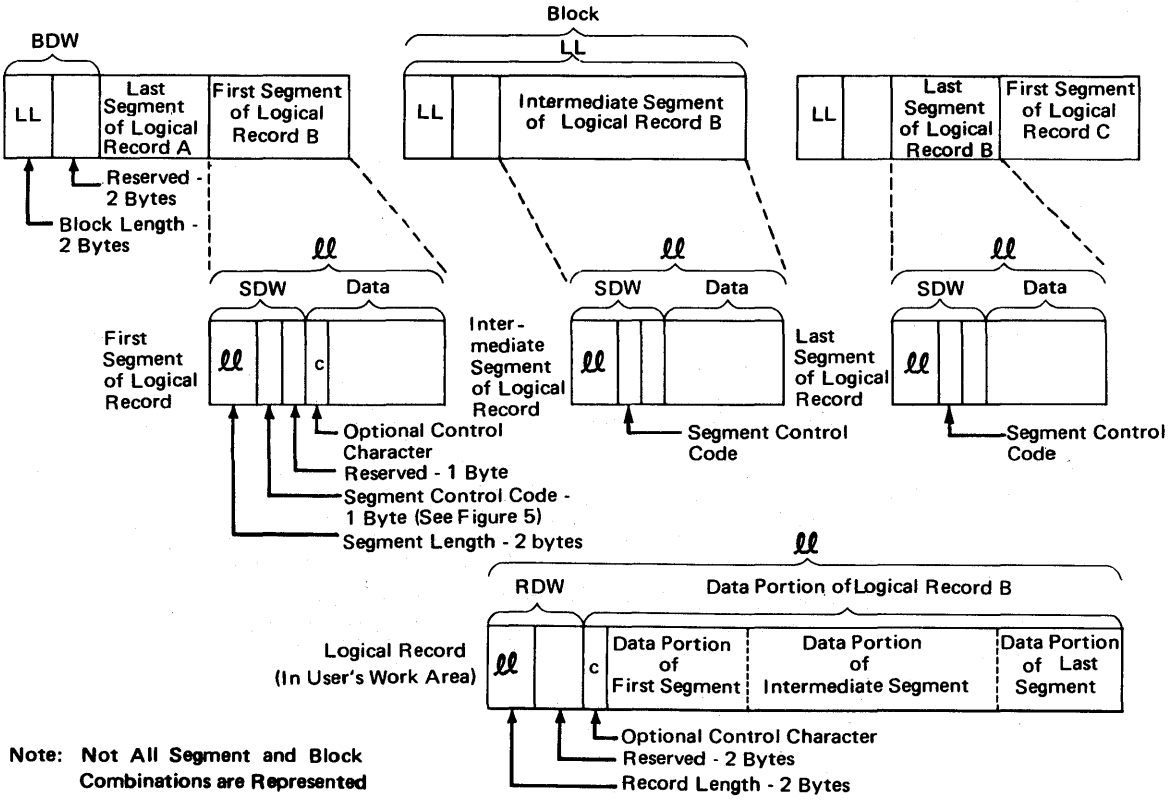


Figure 4. Spanned Variable–Length Records.

When spanning is specified for blocked records, the system tries to fill all blocks. For unblocked records, a record larger than blocksize is split and written in two or more blocks, each block containing only one record or record segment. Thus the blocksize may be set to the one that is best for a given device or processing situation. It is not restricted by the maximum record length of a data set. A record may, therefore, span several blocks, and may even span volumes. Note that a logical record spanning three or more volumes cannot be processed in update mode (described under "Buffer Control") by QSAM. A block can contain a combination of records and record segments, but not multiple segments of the same record. When records are added to or deleted from a data set, or when the data set is processed again with different blocksize or record–size parameters, the record segmenting will change.

Segment Descriptor Word: Each record segment consists of a segment descriptor word (SDW) followed by the data. The segment descriptor word, similar to the record descriptor word, is a 4–byte field that describes the segment. The first 2 bytes contain the length ('ll') of the segment, including the 4–byte SDW. The length can be from 5 to 32,756 bytes or, when you are using WRITE with tape, from 18 to 32,756 bytes. The third byte of the SDW contains the segment control code, which specifies the relative position of the segment in the logical record. The segment control code is in the rightmost 2 bits of the byte. The segment control codes are shown in Figure 5. The remaining bits of the third byte and all of the fourth byte are reserved for future system use and must be 0.

Binary Code	Relative Position of Segment
00	Complete logical record
01	First segment of a multisegment record
10	Last segment of a multisegment record
11	Segment of a multisegment record other than the first

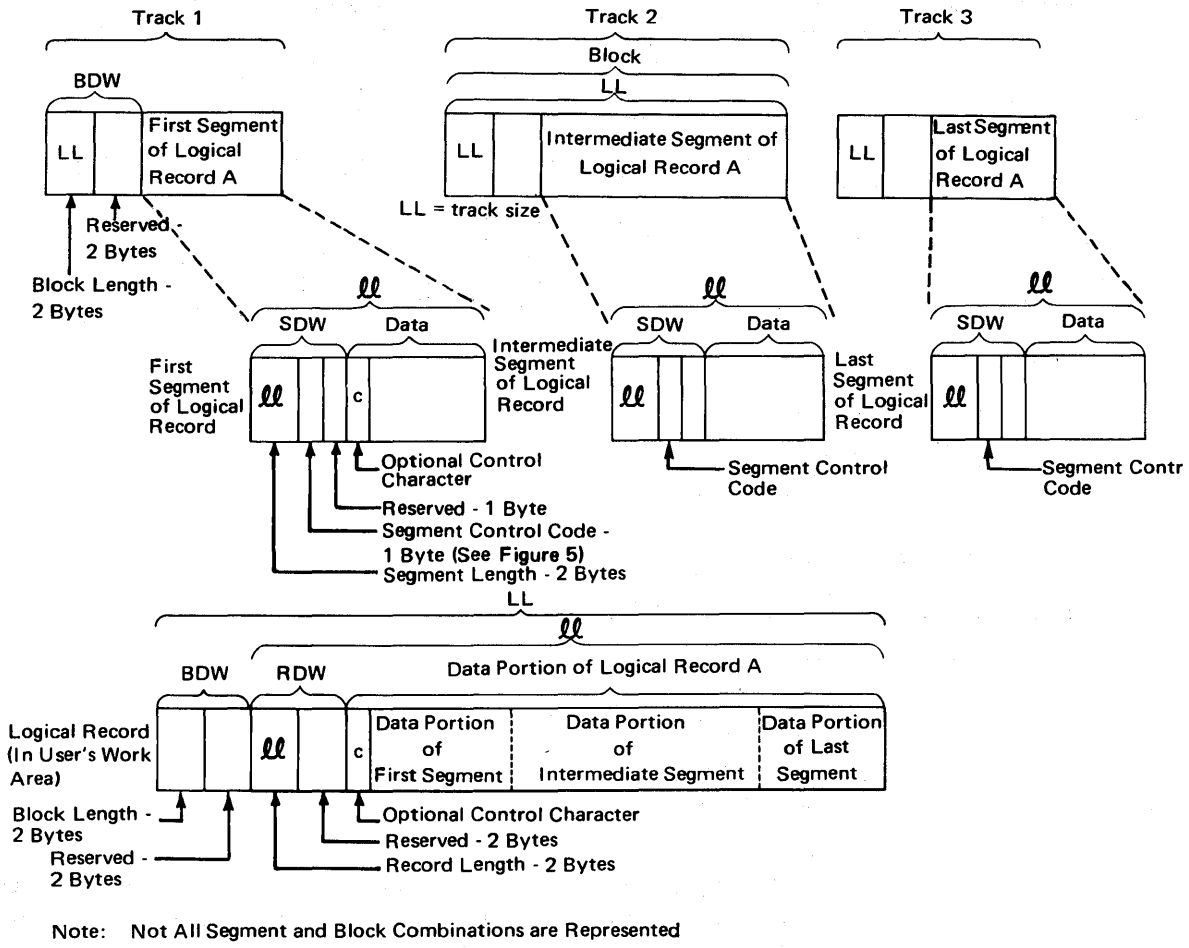
Figure 5. Segment Control Codes

The SDW for the first segment replaces the RDW for the record after the record has been segmented. You or OS can build the SDW, depending on which mode of processing is used. In the basic sequential access method, you must create and interpret the spanned records yourself. In the queued sequential access method move mode, complete logical records, including the RDW, are processed in your work area. GET consolidates segments into logical records and creates the RDW. PUT forms segments as required and creates the SDW for each segment. Data mode is similar to move mode, but allows reference only to the data portion of the logical record in your work area. The logical record length is passed to you through the DCBLRECL field of data control block. In locate mode, both GET and PUT process one segment at a time. However, in locate mode, if you provide your own record area using the BUILDRCD macro instruction or if you ask the system to provide a record area by specifying BFTEK=A, then GET, PUT, and PUTX process one logical record at a time.

A logical record spanning three or more volumes cannot be processed when the data set is opened for update.

When unit–record devices are used with spanned records, the system assumes unblocked records and the blocksize must be equivalent to the length of one print line or one card. Records that span blocks are written one segment at a time.

Spanned Variable–Length Records (Basic Direct Access Method): The spanning feature of the basic direct access method (BDAM) enables you to create and process variable–length unblocked logical records that are longer than 1 track. The feature also enables you to pack tracks with variable–length records by splitting the records into segments. These segments can then be written onto more than one track, as shown in Figure 6.



Note: Not All Segment and Block Combinations are Represented

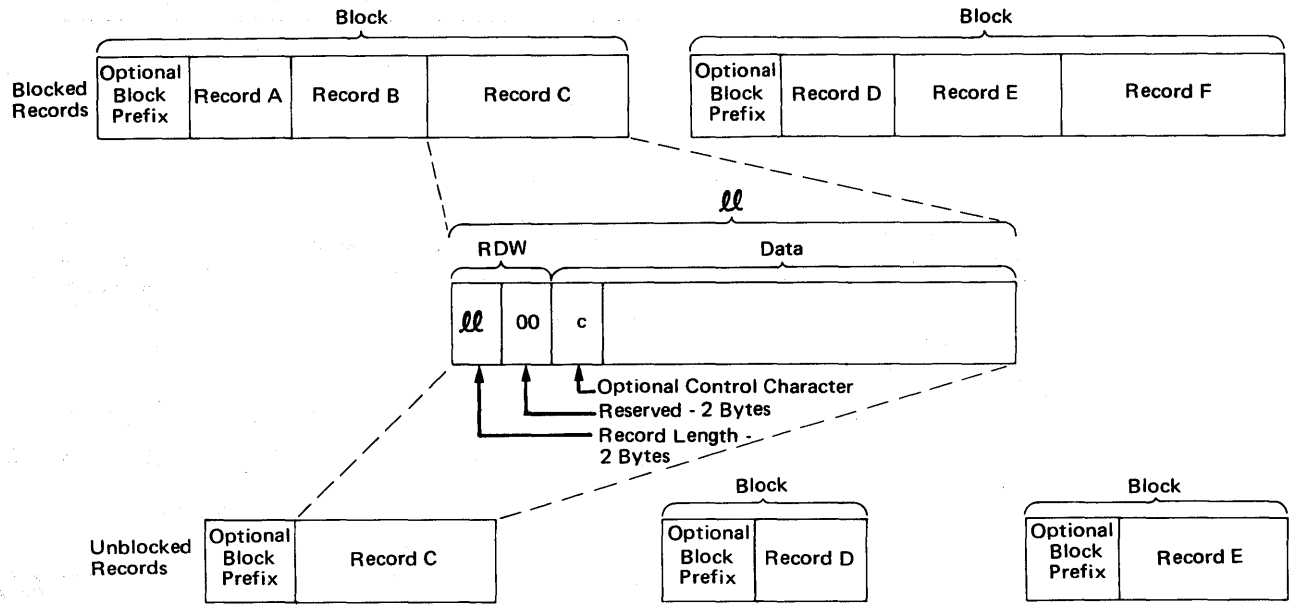
Figure 6. Spanned Variable–Length Records for BDAM Data Sets

When you specify spanned, unblocked record format for the basic direct access method and when a complete logical record cannot fit on the track, the system tries to fill the track with a record segment. Thus the maximum record length of a data set is not restricted by blocksize. Furthermore, segmenting records allows a record to span several tracks, with each segment of the record on a different track. However, since the system does not allow a record to span volumes, all segments of a logical record in a direct data set are on the same volume.

Variable–Length Records — Format D: For ASCII tapes, variable–length records must be format–D records. Format–D records are the same as format–V records, except:

- Control characters, if present, must be ANSI control characters.
- Records or blocks of records can contain block prefixes.

Figure 7 shows the format of variable–length records for ASCII tapes, where the record descriptor word (RDW) must go, and where block prefixes and control characters can go when they exist.



Note: Block prefixes on output records must be 4 bytes long.

Figure 7. Variable–Length Records for ASCII Tapes

The block prefix can vary in length from 0 to 99 bytes but its length must remain constant for the data set being processed. For blocked records, the block prefix precedes the first logical record in each block. For unblocked records, the block prefix precedes each logical record. If the block prefix exists, it precedes the RDW.

To include block prefixes in format–D records on output, code BUFOFF=L as a DCB operand. Your block prefix must be 4 bytes long, and it must contain the length of the block, including the block prefix. If you use QSAM to write records, data management fills in the block prefix for you. If you use BSAM to write records, you must fill in the block prefix yourself. When BUFOFF=L is specified for format–D records, the block prefix is treated as a block descriptor word (BDW) by data management.

When using QSAM, you cannot read the block prefix on input. When using BSAM, you must account for the block prefix on both input and output. When using either QSAM or BSAM, you must account for the length of the block prefix in the BLKSIZE and BUFL operands.

When you use BSAM on output records, the operating system does not recognize the block prefix. Therefore, if you want a block prefix, it must be part of your record.

The block prefix can contain any data you want, but you must avoid using data types, such as binary, packed decimal, and floating–point, that cannot be translated into

ASCII. For format–D records, the only time the block prefix can contain binary data is when you have coded BUFOFF=L, which tells data management that the prefix is a BDW. Unlike the block prefix, the RDW must always be in binary.

If you create variable–length records that are shorter than 18 bytes, data management pads each one up to a length of 18 bytes when the records are written onto ASCII tape. The padding character used is the ASCII circumflex.

For more information about control characters, refer to "Control Character" and to "Appendix B: Control Characters."

Undefined–Length Records

Format U permits processing of records that do not conform to the F or V format. As shown in Figure 8, each block is treated as a record; therefore, deblocking must be performed by your program. The optional control character may be used in the first byte of each record. Because the system does not perform length checking on format–U records, your program may be designed to read less than a complete block into main storage.

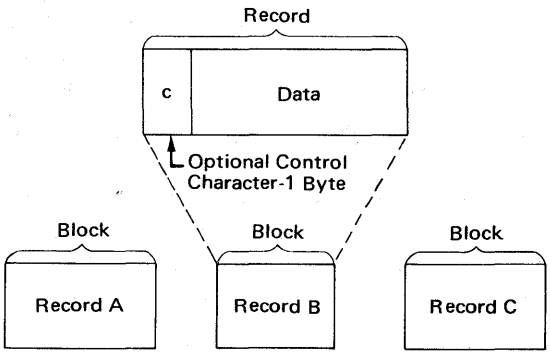


Figure 8. Undefined–Length Records

For ASCII tapes, format–U records are the same as described above, with the two exceptions described for format–F records on ASCII tapes.

Figure 9 shows the format of undefined–length records for ASCII tapes and where control characters and blocks prefixes, if any, go.

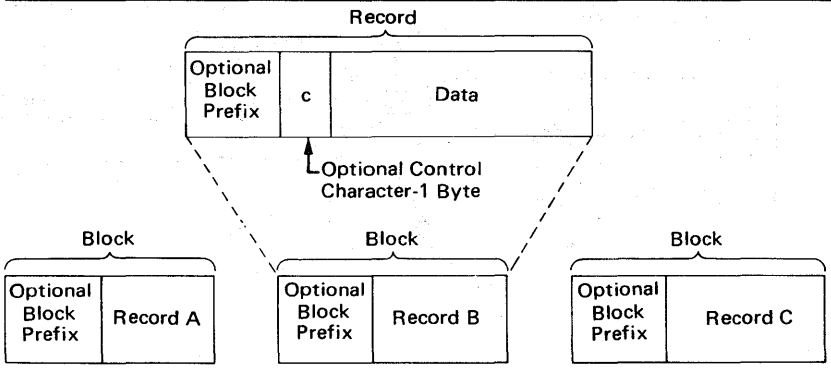


Figure 9. Undefined–Length Records for ASCII Tapes

Control Character

You may specify in the DD statement, the DCB macro instruction, or the data set label that an optional control character is part of each record in the data set. The 1–byte character is used to indicate a carriage control channel when the data set is printed or a stacker bin when the data set is punched. Although the character is a part of the record in storage, it is never printed or punched. For that reason, buffer areas must be large enough to accommodate the character. If the immediate destination of the record is a device, such as disk, that does not recognize the control character, the system assumes that the control character is the first byte of the data portion of the record. If the destination of the record is a printer or punch and you have not indicated the presence of a control character, the system regards the control character as the first byte of data. A list of the control characters is in "Appendix B: Control Characters."

Direct–Access Device Characteristics

Regardless of organization, data sets created using the operating system can be stored on a direct–access volume. Each block of data has a distinct location and a unique address, making it possible to locate any record without extensive searching. Thus, records can be stored and retrieved either directly or sequentially.

Although direct–access devices differ in physical appearance, capacity, and speed, they are similar in data recording, data checking, data format, and programming. The recording surface of each volume is divided into many concentric *tracks.* The number of tracks and their capacity vary with the device. Each device has some type of *access mechanism,* containing read/write heads that transfer data as the recording surface rotates past them. Only one head at a time can transfer data.

The logical arrangement of related tracks is vertical rather than horizontal. As shown in Figure 10, a cylinder of a 1316 disk pack is composed of 10 tracks, one for each

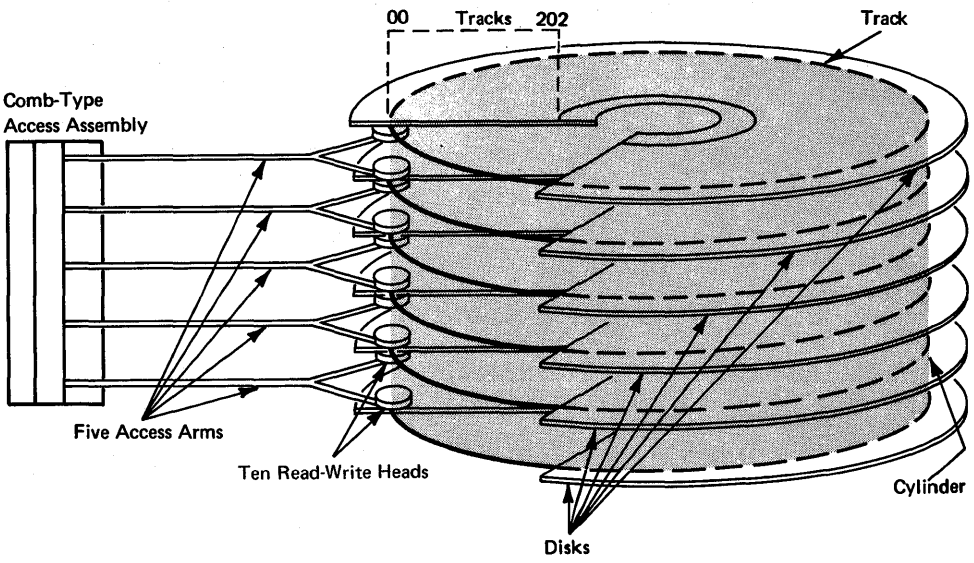


Figure 10. 1316 Disk Pack

recording surface. Because there are 203 tracks per recording surface, there are 203 vertical cylinders of 10 tracks each. If a data set extends to more than 1 track, it is continued on the next track in the cylinder, not the next track on the same recording surface.

Track Format

Information is recorded on all direct–access volumes in a standard format. In addition to device data, each track contains a track descriptor record (*capacity record* or R0) and data records.

As shown in Figure 11, there are two possible data record formats — count–data and count–key–data — only one of which can be used for a particular data set.

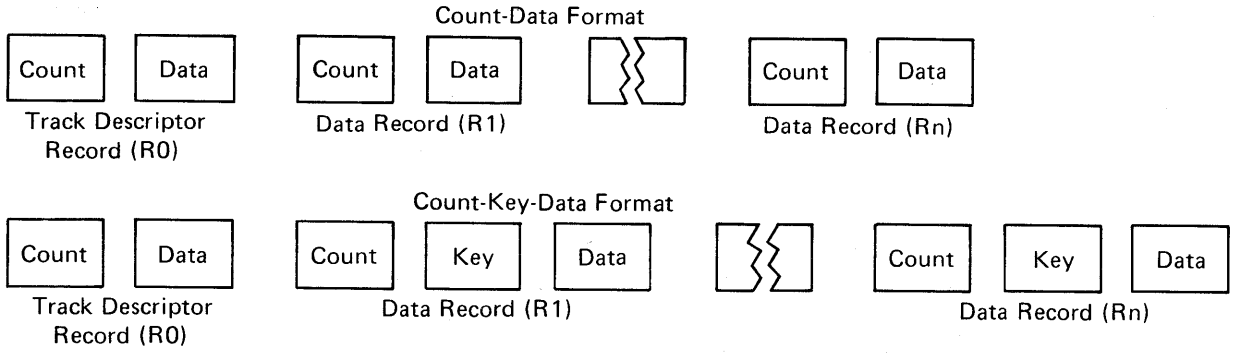


Figure 11. Direct–Access Volume Track Formats

In addition to device data, the count area contains 8 bytes that identify the location of the record by cylinder, head, and record numbers, its key length (0 if no keys are used), and its data length.

If the records are written with keys, the key area (1 to 255 bytes) contains a record key that specifies the data record by part number, account number, sequence number, or some other identifier. In some cases, records are written with keys so that they can be located quickly.

Track Addressing

Two types of addresses can be used to store and retrieve data on a direct–access volume: actual addresses and relative addresses. The only advantage of using actual addresses is the elimination of time required to convert from relative to actual addresses and vice versa. When sequentially processing a multiple–volume data set, you can refer only to records of the current volume.

Actual Addresses: When the system returns the actual address of a record on a direct–access volume to your program, it is in the form MBBCCHHR, where:

M

is a 1–byte binary number specifying the relative location of an entry in a data extent block (DEB). The data extent block is created by the system when the data set is opened. Each extent entry describes a set of consecutive tracks allocated for the data set.

BBCCHH

is three 2–byte binary numbers specifying the cell (bin), cylinder, and head number for the record (its track address). The cylinder and head numbers are recorded in the count area for each record.

R

is a 1–byte binary number specifying the relative block number on the track. The block number is also recorded in the count area.

If you use actual addresses in your program, the data set must be treated as unmovable.

Relative Addresses: Two kinds of relative addresses can be used to refer to records in a direct–access data set: relative block addresses and relative track addresses.

The relative block address is a 3–byte binary number that indicates the position of the block relative to the first block of the data set. Allocation of noncontinuous sets of blocks does not affect the number. The first block of a data set always has a relative block address of 0.

The relative track address has the form TTR, where:

TT

is a 2–byte binary number specifying the position of the track relative to the first track allocated for the data set. The TT for the first track is 0. Allocation of noncontinuous sets of tracks does not affect the number.

R

is a 1–byte binary number specifying the number of the block relative to the first block on the track TT. The R value for the first block of data on a track is 1.

Track Overflow

If the record overflow feature is available for the direct–access device being used, you can reduce the amount of unused space on the volume by specifying the *track overflow option* in the DD statement or the DCB macro instruction associated with the data set. If the option is used, a block that does not fit on the track is partially written on that track and continued on the next track. (The track onto which the record is continued must be physically next and must be part of the same extent as the track that holds the first part of the record.) Each segment (the portion written on one track) of an overflow block has a count area. The data length field in the count area specifies the length of that segment only. If the block is written with a key, there is only one key area for the block. It is written with the first segment. If the track overflow option is not used, blocks are not split between tracks.

Write–Validity–Check Option

You can specify the *write–validity–check option* in either the DD statement or the DCB macro instruction. After a record is transferred from main to secondary storage, the system reads the stored record (without data transfer) and, by testing for a data check from the I/O device, verifies that the record was written correctly. This verification requires an additional revolution of the device for each record that was written. Standard error recovery procedures are initiated if an error condition is detected.

The Data Control Block

You must describe the characteristics of a data set, the volume on which it resides, and its processing requirements before processing can begin. During execution, the descriptive information is made available to the operating system in the *data control block (DCB).* A DCB is required for each data set and is created in a processing program by a DCB macro instruction.

Primary sources of information to be placed in the data control block are a DCB macro instruction, a data definition (DD) statement, and a data set label. In addition, you can provide or modify some of the information during execution by storing the pertinent data in the appropriate field of the data control block. The specifications needed for input/output operations are supplied during the initialization procedures of the OPEN macro instruction. Therefore, the pertinent data can be provided when your job is to be executed rather than when you write your program (see Figure 12).

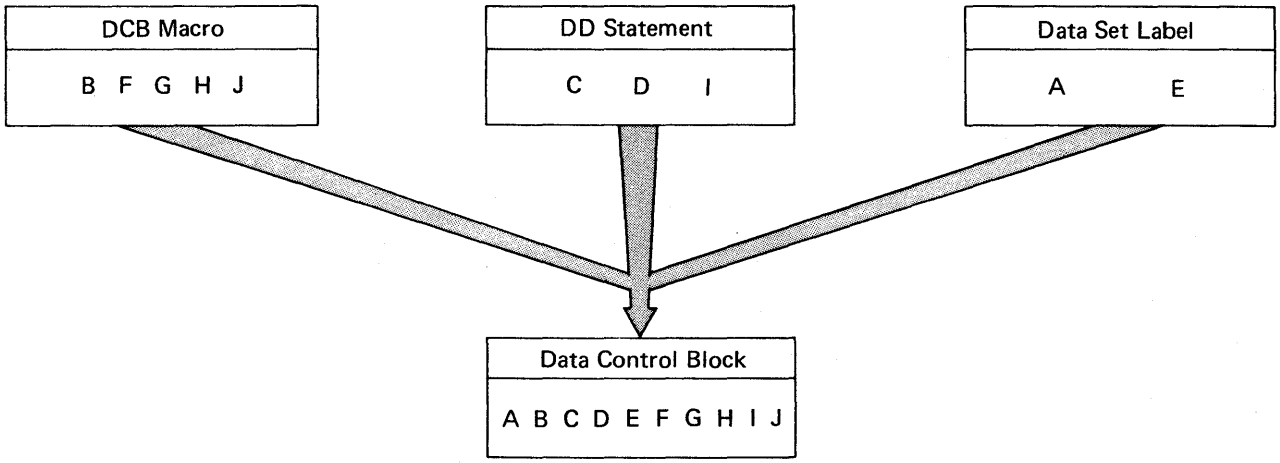


Figure 12. Completing the Data Control Block

When the OPEN macro instruction is executed, the Open routine:

- Completes the data control block
- Loads all necessary data access routines not already in main storage
- Initializes data sets by reading or writing labels and control information
- Constructs the necessary system control blocks

Information from a DD statement is stored in the *job file control block (JFCB)* by the operating system. When the job is to be executed, the JFCB is made available to the Open routine. The data control block is filled in with information from the DCB macro instruction, the JFCB, or an existing data set label. If more than one source specifies information for a particular field, only one source is used. A DD statement takes precedence over a data set label, and a DCB macro instruction over both. However, you can modify any data control block field either before the data set is opened, or when OS returns control to your program (at the data control block exit). Some fields can be modified during processing.

Figure 13 illustrates the process and the sequence of filling in the data control block from various sources. The primary source is your program, that is, the DCB macro instruction. In general, you should use only those DCB parameters that are needed to ensure correct processing. The other parameters can be filled in when your program is to be executed. When a data set is opened, any field in the JFCB not completed by a DD statement is filled in from the data set label (if one exists). Then any field not completed in the DCB is filled in from the JFCB. Any field in the DCB can then be completed or modified by your own DCB exit routine.

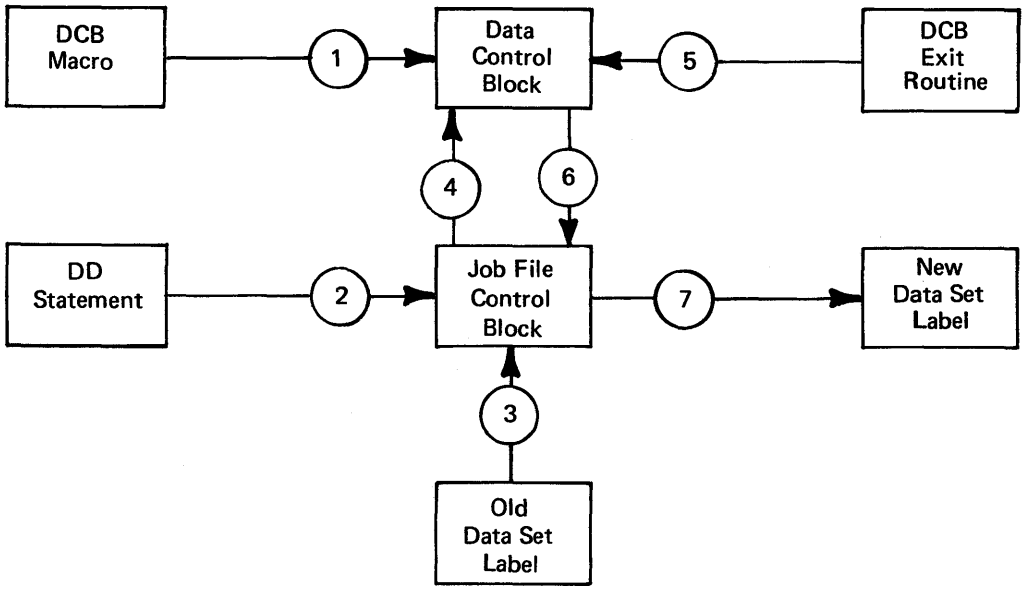


Figure 13. Sources and Sequence of Operations for Completing the Data Control Block

When the data set is closed, the data control block is restored to the condition it had before the data set was opened; it is then available for reuse with another data set. The Open and Close routines also use the updated JFCB to write the data set labels for output data sets. If the data set is not closed when your job terminates, the operating system will close it automatically. Note, however, that the system cannot automatically close any open data sets after the normal termination of a program that was brought into main storage by the loader. Therefore, loaded programs must include CLOSE macro instructions for all opened data sets.

Data Set Description

For each data set you are going to process, there must be a corresponding DCB and DD statement. The characteristics of the data set and device–dependent information can be supplied by either source. In addition, the DD statement must supply data set identification, device characteristics, space allocation requests, and related information as specified in "The DD Statement" in *OS Job Control Language Reference.* You establish the logical connection between a DCB and a DD statement by specifying the name of the DD statement in the DDNAME field of the DCB macro instruction, or by completing the field yourself before opening the data set.

Once the data set characteristics have been specified in the DCB macro instruction, they can be changed only by modification of the DCB during execution. The fields of the DCB discussed below are common to most data organizations and access techniques.

Data Set Organization (DSORG): specifies the organization of the data set as physical sequential (PS), indexed sequential (IS), partitioned (PO), or direct (DA). If the data set contains absolute rather than relative addresses, you must mark it as unmovable by adding a U to the DSORG parameter (for example, by coding DSORG=PSU). You must specify the data set organization in the DCB macro instruction. When creating or processing an indexed sequential organization data set or creating a direct data set, you must also specify DSORG in the DD statement.

Record Format (RECFM): specifies the characteristics of the records in the data set as fixed–length (F), variable–length (V), or undefined–length (U). Blocked records are specified as FB or VB. You may also specify the records as fixed–length standard by using FS or FBS. You can request track overflow for records other than standard format by adding a T to the RECFM parameter (for example, by coding FBT).

Record Length (LRECL): specifies the length, in bytes, of each record in the data set. If the records are of variable length, the maximum record length must be specified. For input, the field should be omitted for format–U records.

Blocksize (BLKSIZE): specifies the maximum length, in bytes, of a block. If the records are of format F, the blocksize must be an integral multiple of the record length except for SYSOUT data sets. (See "Routing Data Sets through the Output Stream" in Part 3 of this book.) If the records are of format V, the blocksize specified must be the maximum blocksize. If records are unblocked, the blocksize must be 4 bytes greater than the record length (LRECL). When spanned variable–length records are specified, the blocksize is independent of the record length.

Each of the data set description fields of the data control block, except as noted for data set organization, can be specified when your job is to be executed. In addition, data set identification and disposition, as well as device characteristics, can be specified at that time. The parameters of the DD statement discussed below are common to most data set organizations and devices.

Data Definition Name (DDNAME): is the name of the DD statement and connects the DD statement to the data control block that specifies the same DDNAME.

Data Set Name (DSNAME): specifies the name of a newly defined data set, or refers to a previously defined data set.

Data Control Block (DCB): provides, by means of subparameters, information to be used to complete those fields of the data control block that were not specified in the DCB macro instruction. This parameter cannot be used to modify a data control block.

Channel Separation and Affinity (SEP/AFF): requests that specified data sets use different channels during input/output operations.

Input/Output Device (UNIT): specifies the number and type of I/O devices to be allocated for use by the data set.

Space Allocation (SPACE): designates the amount of space on a direct–access volume that should be allocated for the data set. Unused space can be released when your job is finished.

Volume Identification (VOLUME): identifies the particular volume or volumes, or the number of volumes, to be assigned to the data set, or the volumes on which existing data sets reside.

Data Set Label (LABEL): indicates the type and contents of the label or labels associated with the data set. The operating system verifies standard labels (SL) and standard user labels (SUL). Nonstandard labels (NSL) can be specified only if your installation has incorporated into the operating system routines to write and process nonstandard labels.

Data Set Disposition (DISP): describes the status of a data set and indicates what is to be done with it at the end of the job step.

Processing Program Description

OS requires several types of processing information to ensure proper control of your input/output operations. The forms of macro instructions in the program, buffering requirements, and the addresses of your special processing routines must be specified during either the assembly or the execution of your program. The DCB parameters specifying buffer requirements are discussed in "Buffer Acquisition and Control."

Because macro instructions are expanded during the assembly of your program, you must supply the macro instruction forms that are to be used in processing each data set in the associated DCB macro instruction. You can supply buffering requirements and related information in the DCB macro instruction, the DD statement, or by storing the pertinent data in the appropriate field of the data control block before the end of your DCB exit routine. If the addresses of special processing routines are omitted from the DCB macro instruction, you must complete them in the DCB before opening the data set.

Macro Instruction Form (MACRF)

The MACRF parameter of the DCB macro instruction specifies not only the macro instructions used in your program, but also the processing mode as discussed in the section "Buffer Control." The organization of your data set, the macro instruction form, and the processing mode determine which of the data access routines will be used during execution.

Exits to Special Processing Routines

The DCB macro instruction can be used to identify the location of:

- A routine that performs end–of–data procedures
- A routine that supplements the operating system's error recovery routine
- A list that contains addresses of special exit routines

The exit addresses can be specified in the DCB macro instruction or you can complete the DCB fields before opening the data set. Figure 14 summarizes the exits that you can specify either explicitly in the DCB, or implicitly by specifying the address of an exit list in the DCB.

Exit Routine	When Available	Where Specified
End–of–Data–Set	When no more sequential records or blocks are available	EODAD operand
Error Analysis	After an uncorrectable input/output error	SYNAD operand
Standard User Label (physical sequential or direct organization)	When opening, closing, or reaching the end of a data set, and when changing volumes	EXLST operand and exit list
Data Control Block	When opening a data set	EXLST operand and exit list
End–of–Volume	When changing volumes	EXLST operand and exit list
Block Count	After unequal block count comparison by end–of–volume routine	EXLST operand and exit list
FCB Image	When opening a data set or issuing a SETPRT macro	EXLST operand and exit list
DCB ABEND	When an ABEND condition occurs in Open, Close, or end–of–volume routine.	EXLST operand and exit list

Figure 14. Data Management Exit Routines

End–of–Data–Set Exit Routine (EODAD): The EODAD parameter of the DCB macro instruction specifies the address of your end–of–data routine, which performs any final processing on an input data set. This routine is entered when a READ or GET request is made and there are no more records or blocks to be retrieved. (On a READ request, the routine is entered when you issue a CHECK macro instruction to check for completion of the read operation.) Your routine can reposition the volume for continued processing (if the access method is BPAM), close the data set, or process the next sequential data set. Under no condition should you issue another QSAM GET request after the data set has encountered the end–of–data condition. If no exit routine is provided, the task will be abnormally terminated.

Synchronous Error Routine Exit (SYNAD): The SYNAD parameter of the DCB macro instruction specifies the address of an error routine that is to be given control when an input/output error occurs. This routine can be used to analyze exceptional conditions or uncorrectable errors. The block being read or written can be accepted or skipped, or processing can be terminated.

If an input/output error occurs during data transmission, standard error recovery procedures, provided by OS, attempt to correct the error before returning control to your program. An uncorrectable error usually causes an abnormal termination of the task. However, if you specify in the DCB macro instruction the address of an error analysis routine (called a SYNAD routine), the routine is given control in the event of an uncorrectable error.

You can write a SYNAD routine to determine the cause and type of error that occurred by examining:

- The contents of the general registers
- The data event control block (discussed in Part 2 under "Basic Access Technique")
- The exceptional condition code
- The standard status and sense indicators

You can use the SYNADAF macro instruction to perform this analysis automatically. This macro instruction produces an error message that can be printed by a subsequent PUT or WRITE macro instruction.

After completing the analysis, you can return control to OS or close the data set. To continue processing the same data set, you must first return control to the control program by a RETURN macro instruction. The control program then transfers control to your processing program, subject to the conditions described below. In no case should you attempt to reread or rewrite the record, because the system has already attempted to recover from the error.

When you are using GET and PUT to process a sequential data set, the operating system provides three automatic error options (EROPT) to be used if there is no SYNAD routine or if you want to return control to your program from the SYNAD routine:

- ACC accept the erroneous block
- SKP skip the erroneous block
- ABE abnormally terminate the task

These options are applicable only to data errors, as control errors result in abnormal termination of the task. Data errors affect only the validity of a block of data. Control errors affect information or operations necessary for continued processing of the data set. These options are not applicable to output errors, except output errors on the printer. When chained scheduling is used, the SKP option is not available, and ACC is assumed if SKP is coded. If the EROPT and SYNAD fields are not completed, ABE is assumed.

When you use READ and WRITE macro instructions, errors are detected when you issue a CHECK macro instruction. If you are processing a direct or sequential data set and you return to the control program from your SYNAD routine, the operating system assumes that you have accepted the bad record. If you are creating a direct data set and you return to the control program from your SYNAD routine, your task is abnormally terminated.

For a detailed description of the register contents upon entry to your SYNAD routine, refer to the tables in the *OS Data Management Macro Instructions* manual. The tables there describe register contents for programs using QISAM, BISAM, BDAM, BPAM, BSAM, and QSAM.

Your SYNAD routine can end by branching to another routine in your program, such as a routine that closes the data set. It can also end by returning control to the control program, which then returns control to the next sequential instruction (after the macro)

in your program. If your routine returns control, the conventions for saving and restoring register contents are as follows:

- The SYNAD routine must preserve the contents of registers 13 and 14. If required by the logic of your program, the routine must also preserve the contents of registers 2 through 12. Upon return to your program, the contents of registers 2 through 12 will be the same as upon return to the control program from the SYNAD routine.
- The SYNAD routine must not use the save area whose address is in register 13, because this area is used by the control program. If the routine saves and restores register contents, it must provide its own save area.
- If the SYNAD routine calls another routine or issues supervisor or data management macro instructions, it must provide its own save area or issue a SYNADAF macro instruction. The SYNADAF macro instruction provides a save area for its own use, and then makes this area available to the SYNAD routine. Such a save area must be removed from the save area chain by a SYNADRLS macro instruction before control is returned to the control program.

When you use QSAM to read and translate paper–tape characters, your SYNAD routine receives control when you request the record preceding the record in error. Before giving control to your SYNAD routine, the system translates the requested record into your buffer.

For example, suppose that you are using QSAM to read and translate a paper–tape data set and that you have specified, in your DCB, SYNAD=(address) and EROPT=ACC. Suppose also that the third record of the data set has a parity error. When you issue a GET request for the second record, the system translates that record into your buffer and, as a result of the error in the third record, passes control to your SYNAD routine. Because you specified the accept option, the system returns control to your program after your SYNAD error analysis routine completes its processing. When you issue a GET request for the third record, all characters other than the erroneous one are translated into your buffer; the erroneous character is moved, in normal sequence, into your buffer without translation.

Exit List (EXLST): The EXLST parameter of the DCB macro instruction specifies the address of a list that contains the addresses of special processing routines, a forms control buffer (FCB) image, or a user totaling area. An exit list must be created if user label, data control block, end of volume, block count, or DCB ABEND exits are used or if an FCB image is defined in the processing program.

The exit list is constructed of 4–byte entries that must be aligned on fullword boundaries. Each exit list entry is identified by a code in the high–order byte, and the address of the routine, image, or area is specified in the 3 low–order bytes. Codes and addresses for the exit list entries are shown in Figure 15.

You can activate or deactivate any entry in the list by placing the required code in the high–order byte. Care must be taken, however, so as not to destroy the last entry indication. OS scans the list from top to bottom, and the first active entry found with the proper code is selected.

You can shorten the list during execution by setting the high–order bit to 1, and extend it by setting the high–order bit to 0.

Entry Type	Hexadecimal Code	3–Byte Address — Purpose
Inactive entry	00	Ignore the entry; it is not active.
Input header label	01	Process a user input header label.
Output header label	02	Create a user output header label.
Input trailer label	03	Process a user input trailer label.
Output trailer label	04	Create a user output trailer label.
Data control block exit	05	Take a data control block exit.
End–of–volume	06	Take an end–of–volume exit.
User totaling	0A	Assume this address is beginning of user's totaling area.
Block count exit	0B	Take a block–count–unequal exit.
Defer input trailer label	0C	Defer processing of a user input trailer label from end–of–data until closing (no exit routine address).
Defer nonstandard input trailer label	0D	Defer processing a nonstandard input trailer label on magnetic tape unit from end–of–data until closing (no exit routine address).
FCB image	10	Define an FCB image.
DCB ABEND exit	11	Examine the ABEND condition and select one of several options.
Last entry	80	Treat this entry as last entry in list. This code can be specified with any of the above but must always be specified with the last entry.

Figure 15. Format and Contents of an Exit List

When control is passed to an exit routine, the registers contain the following information:

Register	Contents
0	Variable; see exit routine description
1	Address of data control block currently being processed
2–13	Contents before execution of the macro instruction
14	Return address (must not be altered by the exit routine)
15	Address of exit routine entry point

The conventions for saving and restoring register contents are as follows:

- The exit routine must preserve the contents of register 14. It need not preserve the contents of other registers. The control program restores the contents of registers 2–13 before returning control to your program.
- The exit routine must not use the save area whose address is in register 13, because this area is used by the control program. If the exit routine calls another routine or issues supervisor or data management macro instructions, it must provide the address of a new save area in register 13.

Standard User Label Exit: When you create a data set with physical sequential or direct organization, you can provide routines to create your own data set labels. You can also provide routines to verify these labels when you use the data set as input. Each label is 80 characters long with the first 4 characters UHL1,UHL2,...,UHL8 for a header label or UTL1,UTL2,...,UTL8 for a trailer label.

The physical location of the labels on the data set depends on the data set organization. For direct (BDAM) data sets, user labels are placed on a separate user label track in the first volume. User label exits are taken only during execution of the Open and Close routines. Thus you may create or examine up to eight user header labels only during execution of Open and up to eight trailer labels only during execution of Close. Since the trailer labels are on the same track as the header labels, the first volume of the data set must be mounted when the data set is closed. For physical sequential (BSAM or QSAM) data sets, you may create or examine up to eight header labels and eight trailer labels on each volume of the data set. For ASCII tape data sets, you may create unlimited user header and trailer labels. The user label exits are taken during open, close, and end–of–volume processing.

To create or verify labels, you must specify the addresses of your label exit routines in an exit list for use during standard label processing. Thus you may have separate routines for creating or verifying header and trailer label groups. Care must be taken if a magnetic tape is read backward, since the trailer label group is processed as header labels and the header label group is processed as trailer labels.

When your routine receives control, the contents of register 0 are unpredictable. Register 1 contains the address of a parameter list. The contents of registers 2–13 are the same as when the macro instruction was issued. However, if your program does not issue the CLOSE macro instruction, or abnormally terminates before issuing CLOSE, the CLOSE macro instruction will be issued by the control program, with control–program information in these registers.

The parameter list pointed to by register 1 is a 16–byte area aligned on a fullword boundary. Figure 16 shows the contents of the area.

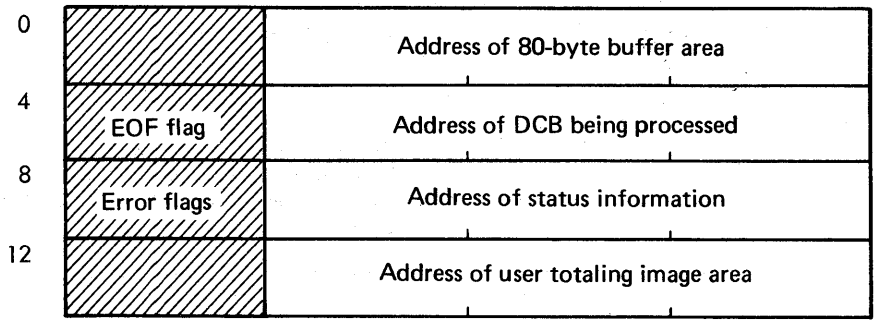


Figure 16. Parameter List Passed to User Label Exit Routine

The first address in the parameter list points to an 80–byte label buffer area. For input, the control program reads a user label into this area before passing control to the label routine. For output, the user label exit routine constructs labels in this area and returns to the control program, which writes the label. When an input trailer label

routine receives control, the EOF flag (high–order byte of the second entry in the parameter list) is set as follows:

bit 0 = 0:	Entered at end–of–volume
bit 0 = 1:	Entered at end–of–file
bits 1–7:	Reserved

When a user label exit routine receives control after an uncorrectable I/O error has occurred, the third entry of the parameter list contains the address of the standard status information. The error flag (high–order byte of the third entry in the parameter list) is set as follows:

bit 0 = 1:	Uncorrectable I/O error
bit 1 = 1:	Error occurred during writing of updated label
bits 2–7:	Reserved

The fourth entry in the parameter list is the address of the user totaling image area. This image area is the entry in the user totaling save area that corresponds to the last record physically written on the volume. The image area is discussed further under "User Totaling."

Each routine must create or verify one label of a header or trailer label group, place a return code in register 15, and return control to the operating system. The operating system responds to the decimal return code as shown in Figure 17.

Routine Type	Return Code	System Response
Input header or trailer label	0	Normal processing is resumed. If there are any remaining labels in the label group, they are ignored.
	4	The next user label is read into the label buffer area and control is returned to the exit routine. If there are no more labels in the label group, normal processing is resumed.
	8	The label is written from the label buffer area and normal processing is resumed.[1]
	12	The label is written from the label area, the next label is read into the label buffer area, and control is returned to the label processing routine. If there are no more labels, processing is resumed.[1]
Output header or trailer label	0	Normal processing is resumed; no label is written from the label buffer area.
	4	User label is written from the label buffer area. Normal processing is resumed.
	8	User label is written from the label buffer area. If fewer than eight labels have been created, control is returned to the exit routine, which then creates the next label. If eight labels have been created, normal processing is resumed.

[1] Only for a physical sequential data set opened for update or a direct data set opened for update or output.

Figure 17. System Response to a User Label Exit Routine Return Code

You can create user labels only for data sets on magnetic–tape volumes with IBM standard labels or American National Standard labels and for data sets on direct–access volumes. When you specify both user labels and IBM standard or American National Standard labels in a DD statement by specifying LABEL=(,SUL) or LABEL=(,AUL) and there is an active entry in the exit list, a label exit is always taken. A label exit may be taken when an input data set does not contain user labels, or when no user

label track has been allocated for writing labels on a direct–access volume. In either case, the appropriate exit routine is entered with the buffer area address parameter set to 0. On return from the exit routine, normal processing is resumed; no return code is necessary.

Label exits are not taken for system output (SYSOUT) data sets, or for data sets on volumes that do not have standard labels. For other data sets, exits are taken as follows:

- When the data set is opened, header label exits are taken, except when the data set already exists and DISP=MOD is coded in the DD statement. In the latter case, the volume is positioned to the end of the data set, and input trailer label exits are taken.
- When end–of–volume is reached, trailer label exits are taken; header label exits are taken after volume switching. Input trailer label exits are not taken, however, if you force end–of–volume by issuing an FEOV macro instruction.
- When end–of–data is reached, input trailer label exits are taken before the EODAD exit, unless the DCB exit list indicates defer input trailer label processing. When an output data set is closed, output trailer label exits are taken.
- When end–of–data is reached for a direct–access data set and the DCB exit list indicates that the system is to defer input trailer label processing, the system changes the 0C to 0D. When the Close routine has finished processing, the system changes the code back to 0C.

To process records in reverse order, a data set on magnetic tape can be read backward. When you read backward, header label exits are taken to process trailer labels, and trailer label exits are taken to process header labels. The system presents labels from a label group in ascending order by label number, which is the order in which the labels were created. If necessary, an exit routine can determine label type (UHL or UTL) and number by examining the first four characters of each label. Tapes with IBM standard labels can have as many as eight user labels. Tapes with American National Standard labels can have unlimited user labels.

If an uncorrectable error occurs during reading or writing of a user label, the system passes control to the appropriate exit routine with the third word of the parameter list flagged and pointing to status information.

After an input error, the exit routine must return control with an appropriate return code (0 or 4). No return code is required after an output error. If an output error occurs while the system is opening a data set, the data set is not opened (DCB is flagged) and control is returned to your program. If an output error occurs at any other time, the system attempts to resume normal processing.

A sample program illustrating user label processing is included in SYS1.SAMPLIB. This program, named USERLABL, is discussed in *OS System Generation.*

User Totaling (BSAM and QSAM only): When creating or processing a data set with user labels, you may develop control totals for each volume of the data set and store this information in your user labels. For example, a control total that was accumulated as the data set was created can be stored in your user label and later compared with a total accumulated during processing of the volume. User totaling assists you by synchronizing the control data you create with records physically written on a volume.

For an output data set without user labels, you can also develop a control total that will be available to your end–of–volume routine.

To request user totaling, you must specify OPTCD=T in the DCB macro instruction or in the DCB parameter of the DD statement. The area in which you accumulate the control data (the user totaling area) must be identified to the control program by an entry of hexadecimal 0A in the DCB exit list.

The user totaling area, an area in storage that you provide, must begin on a halfword boundary and be large enough to contain your accumulated data plus a 2–byte length field. The length field must be the first 2 bytes of the area and specify the length of the entire area. A data set for which you have specified user totaling (OPTCD=T) will not be opened if either the totaling area length or the address in the exit list is 0, or if there is no X'0A' entry in the exit list.

The control program establishes a user totaling save area, in which the control program preserves an image of your totaling area, when an I/O operation is scheduled. When the output user label exits are taken, the address of the save area entry (user totaling image area) corresponding to the last record physically written on a volume is passed to you in the fourth entry of the user label parameter list. This parameter list is described in the section "Standard User Label Exit." When an end–of–volume exit is taken for an output data set and user totaling has been specified, the address of the user totaling image area is in register 0.

When using user totaling for an output data set, that is, when creating the data set, you must update your control data in your totaling area before issuing a PUT or a WRITE macro instruction. The control program places an image of your totaling area in the user totaling save area when an I/O operation is scheduled. A pointer to the save area entry (user totaling image area) corresponding to the last record physically written on the volume, is passed to you in your label processing routine. Thus you can include the control total in your user labels. When subsequently using this data set for input, you can accumulate the same information as you read each record and compare this total with the one previously stored in the user trailer label. If you have stored the total from the preceding volume in the user header label of the current volume, you can process each volume of a multivolume data set independently and still maintain this system of control.

When variable–length records are specified with the totaling facility for user labels, special considerations are necessary. Since the control program determines whether a variable–length record will fit in a buffer after a PUT or a WRITE has been issued, the total you have accumulated may include one more record than is actually written on the volume. In the case of variable–length spanned records, the accumulated total will include the control data from the volume–spanning record although only a segment of the record is on that volume. However, when you process such a data set, the volume–spanning record or the first record on the next volume will not be available to you until after the volume switch and user label processing are completed. Thus the totaling information in the user label may not agree with that developed during processing of the volume.

One way you can resolve this situation is to maintain, when you are creating a data set, control data pertaining to each of the last two records and include both totals in your user labels. Then the total related to the last complete record on the volume and the volume–spanning record or the first record on the next volume would be available to your user label routines. During subsequent processing of the data set, your user label

routines can determine if there is agreement between the generated information and one of the two totals previously saved.

Data Control Block Exit: You can specify in an exit list the address of a routine that completes or modifies a DCB and does any additional processing required before the data set is completely open. The routine is entered during the opening process after the JFCB has been used to supply information for the DCB. The routine can determine data set characteristics by examining fields completed from the data set labels.

As with label processing routines, register 14's contents must be preserved and restored if any macro instructions are used in the routine. Control is returned to the operating system by a RETURN macro instruction; no return code is required.

End–of–Volume Exit: You can specify in an exit list the address of a routine that is entered when end–of–volume is reached in processing of a physical sequential data set.

When the end–of–volume routine is entered, register 0 contains 0 unless user totaling was specified. If you specified user totaling in the DCB macro instruction (by coding OPTCD=T) or in the DD statement for an output data set, register 0 contains the address of the user totaling image area. The routine is entered after a new volume has been mounted and all necessary label processing has been completed. If the volume is a reel of magnetic tape, the tape is positioned after the tapemark that precedes the beginning of the data.

You can use the end–of–volume exit routine to take a checkpoint by issuing the CHKPT macro instruction, which is discussed in *OS Supervisor Services and Macro Instructions*. If the job step terminates abnormally, it can be restarted from this checkpoint. When the job step is restarted, the volume is mounted and positioned as upon entry to the routine. Note that restart becomes impossible if changes are subsequently made to the system SVC library (SYS1.SVCLIB). When the step is restarted, pointers to end–of–volume modules must be the same as when the checkpoint was taken.

The end–of–volume exit routine returns control in the same manner as the data control block exit routine. Register 14's contents must be preserved and restored if any macro instructions are used in the routine. Control is returned to the operating system by a RETURN macro instruction; no return code is required.

Block Count Exit: You can specify in an exit list the address of a routine that will allow you to abnormally terminate the task or continue processing when the end–of–volume routine finds an unequal block count condition. When you are using standard labeled input tapes, the block count in the trailer label is compared by the end–of–volume routine with the block count in the DCB. The count in the trailer label reflects the number of blocks written when the data set was created. The number of blocks read when the tape is used as input is contained in the DCBBLKCT field of the DCB.

The routine is entered during end–of–volume processing. The trailer label block count is passed in register 0. You may gain access to the count field in the DCB by using the address passed in register 1 plus the proper displacement, as given in *OS System Control Blocks*. If the block count in the DCB differs from that in the trailer label when no exit routine is provided, the task is abnormally terminated.

The routine must terminate with a RETURN macro instruction and a return code that indicates what action is to be taken by the operating system, as shown in Figure 18. As with other exit routines, register 14's contents must be saved and restored if any macro instructions are used.

Return Code	System Action
0	The task is abnormally terminated.
4	Normal processing is resumed.

Figure 18. System Response to Block Count Exit Return Code

Defer Nonstandard Input Trailer Label Exit: In an exit list, you can specify a code that indicates that you want to defer nonstandard input trailer label processing from end–of–data until the data set is closed. The address portion of the entry is not used by the operating system.

An end–of–volume condition exists in several situations. Two are when the system reads a filemark or tapemark at the end of a volume of a multivolume data set but that volume is not the last, and when the system reads a filemark or tapemark at the end of a data set. The first situation is referred to here as an end–of–volume condition, and the second as an end–of–data condition, although it, too, can occur at the end of a volume.

For an end–of–volume (EOV) condition, the EOV routine passes control to your nonstandard input trailer label routine, whether or not this exit code is specified. For an end–of–data condition when this exit code is specified, the EOV routine does not pass control to your nonstandard input trailer label routine. Instead, the Close routine passes control to your end–of–data routine.

FCB Image Exit: You can specify in an exit list the address of a forms control buffer (FCB) image. This FCB image can be loaded into the forms control buffer of the printer control unit. The FCB controls the movement of forms in printers that do not use a carriage control tape.

The first 4 bytes of the FCB image contain the image identifier. To load the FCB, this image identifier is specified in the FCB parameter of the DD statement or the SETPRT macro instruction.

The image identifier is followed by the FCB load module, described in *OS Data Management for System Programmers.*

You can use an exit list to define an FCB image only when writing to an online printer. Figure 19 illustrates one way the exit list can be used to define an FCB image.

DCB ABEND Exit: You can specify in the DCB exit list the address of your DCB ABEND exit routine to be entered when an ABEND condition occurs during open, close, or end–of–volume processing. When an ABEND condition occurs, a message about the ABEND condition is issued and the DCB ABEND exit routine is given control. The contents of the registers when the exit routine is entered are the same as for other DCB exit list routines except that register 1 contains the address of the parameter list described in Figure 20.

```
          ...
          DCB    ..,EXLST=EXLIST
          ...
EXLST     DS     0F
          DC     X'10'           Flag code for FCB image
          DC     AL3(FCBIMG)     Address of FCB image
          DC     X'80000000'     End of EXLST
FCBIMG    DC     CL4'IMG1'       FCB identifier
          DC     X'00'           FCB is not a default
          DC     AL1(66)         Length of FCB
          DC     X'00'           Spacing is 6 lines per inch
          DC     5X'00'          Lines 3-6 no channel codes
          DC     X'01'           Line 7 channel 1
          DC     6X'00'          Lines 8-13 no channel codes
          DC     X'02'           Line (or Lines) 14 channel 2
          DC     5X'00'          Line (or Lines) 15-19 no channel codes
          DC     X'03'           Line (or Lines) 20 channel 3
          DC     9X'00'          Line (or Lines) 21-29 no channel codes
          DC     X'04'           Line (or Lines) 30 channel 4
          DC     19X'00'         Line (or Lines) 31-49 no channel codes
          DC     X'05'           Line (or Lines) 50 channel 5
          DC     X'06'           Line (or Lines) 51 channel 6
          DC     X'07'           Line (or Lines) 52 channel 7
          DC     X'08'           Line (or Lines) 53 channel 8
          DC     X'09'           Line (or Lines) 54 channel 9
          DC     X'0A'           Line (or Lines) 55 channel 10
          DC     X'0B'           Line (or Lines) 56 channel 11
          DC     X'0C'           Line (or Lines) 57 channel 12
          DC     8X'00'          Line (or Lines) 58-65 no channel codes
          DC     X'10'           End of FCB image
          ...
          END
//ddname  DD UNIT=3211,FCB=(IMG1,VERIFY)
/*
```

Figure 19. Defining an FCB Image

You must inspect bits 4, 5, and 6 of the option mask byte (byte 3 of the parameter list) to determine which options are available. If a bit is set to 1, the corresponding option is available. Indicate your choice by inserting the appropriate value in byte 3 of the parameter list, overlaying the bits you inspected. If you use a value that specifies an option that is not available, the ABEND is issued immediately.

If the contents of the option mask are 0, you must request an immediate ABEND by leaving the value of 0 in the option mask unchanged.

If bit 5 of the option mask is set to 1, you can ignore the ABEND by placing a decimal value of 4 in byte 3 of the parameter list. Processing on the current DCB stops. If you subsequently attempt to use this DCB, the results are unpredictable. If you ignore an error in end–of–volume, the data set will be closed before control is returned to your program.

If bit 6 of the option mask is set to 1, you can delay the ABEND by placing a decimal value of 8 in byte 3 of the parameter list. All other DCBs waiting for open or close processing will be processed before the ABEND is issued. For end–of–volume, however, you can't delay the ABEND because the end–of–volume routine never has more than one DCB to process.

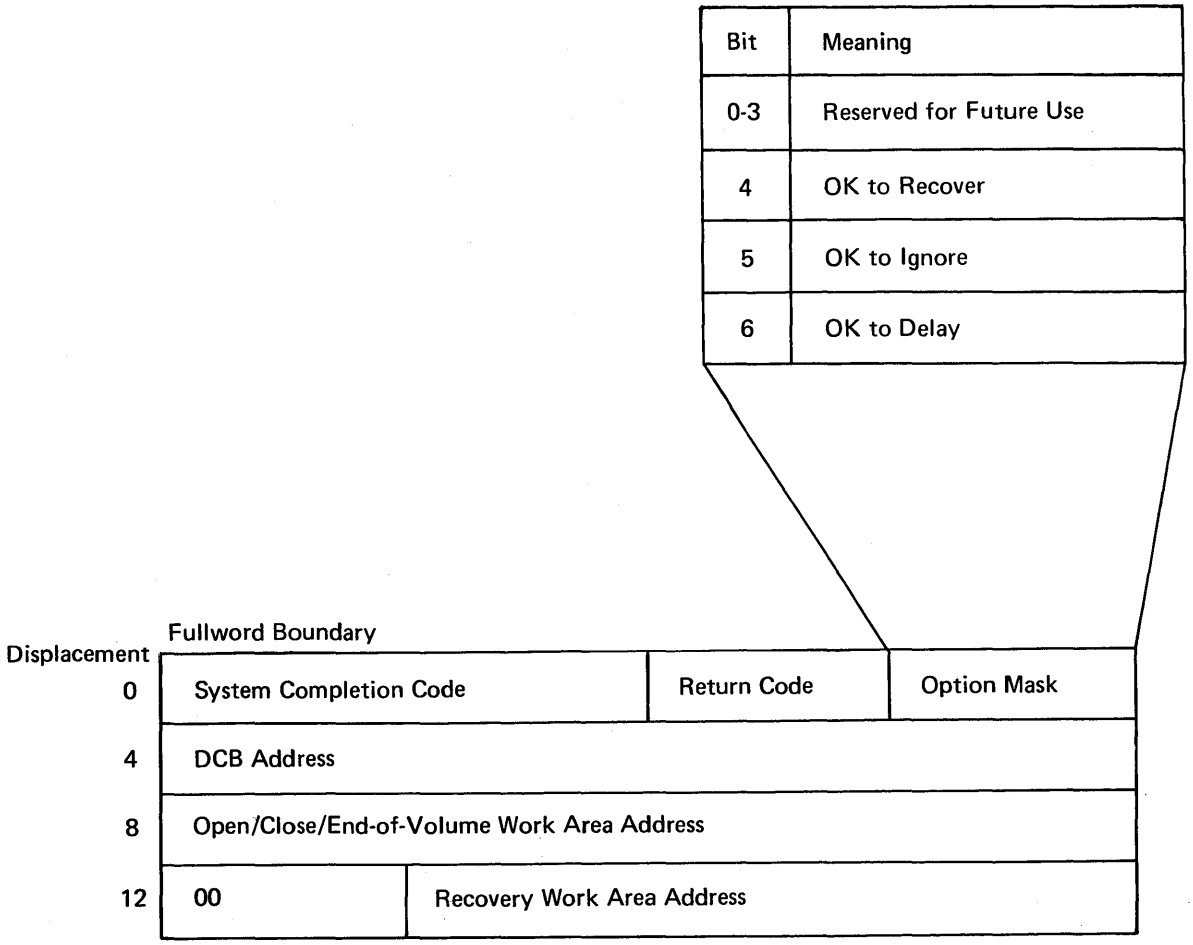


Figure 20. Parameter List Passed to DCB ABEND Exit Routine

If bit 4 of the option mask is set to 1, you can attempt to recover. Place a decimal value of 12 in byte 3 of the parameter list and provide information for the recovery attempt. Figure 21 lists the ABEND conditions for which recovery can be attempted.

For the recovery attempt, you should supply a recovery work area (see Figure 22) with a new volume serial number for each volume associated with an error. If no new volumes are supplied, recovery will be attempted with the existing volumes, but the likelihood of successful recovery is greatly reduced.

If you request recovery for system completion code 213, return code 04, you must indicate in your job control language (JCL) they are nonsharable by specifying unit affinity, deferred mounting, or more volumes than units for the data set.

If you request recovery for system completion code 237, return code 04, you don't need to supply new volumes or a work area. The condition that caused the ABEND is the disagreement between the block count in the DCB and that in the trailer label. This disagreement is ignored to permit recovery.

System Completion Code	Return Code	Description of Error
213	04	DSCB was not found on volume specified.
237	04	Block count in DCB does not agree with block count in trailer label.
413	18	Data set was opened for input and no volume serial number was specified.
613	08	I/O error occurred during reading of tape label.
	0C	Invalid tape label was read.
	10	I/O error occurred during writing of tape label.
	14	I/O error occurred during writing of tapemark following header labels.
717	10	I/O error occurred during reading of trailer label 1 to update block count in DCB.
813	04	Data set name on header label does not match data set name on DD statement.

Figure 21. Conditions for which Recovery Can Be Attempted

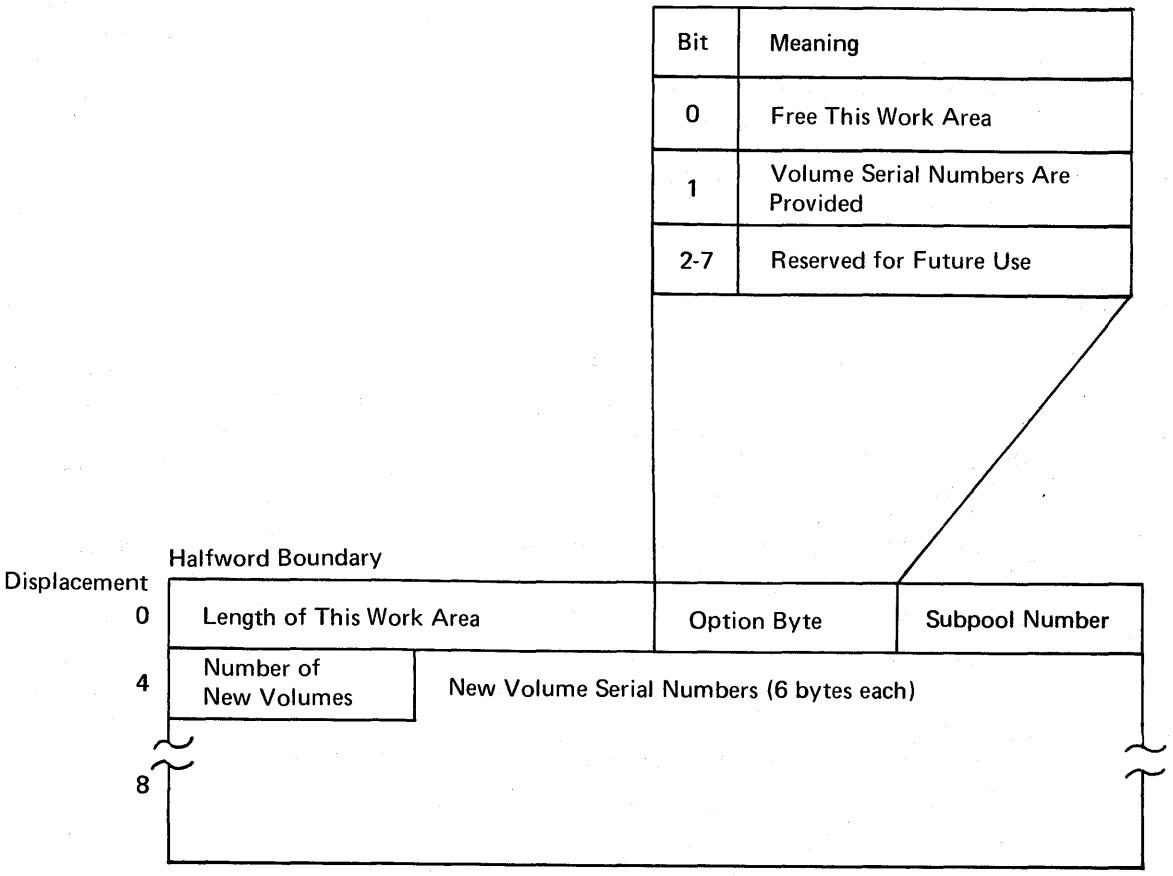


Figure 22. Recovery Work Area

If you request recovery for system completion code 717, return code 10, you don't need to supply new volumes or a work area. The ABEND is caused by an I/O error during updating of the DCB block count. To permit recovery, the block count is not updated. Consequently, an abnormal termination with system completion code 237, return code 04, may result when you try to read from the tape after recovery. You may attempt recovery from the ABEND with system completion code 237, return code 04, as explained in the preceding paragraph.

System completion codes and their return codes are described in *OS Messages and Codes*.

The work area that you supply for the recovery attempt must begin on a halfword boundary and can contain the information described in Figure 22. Place a pointer to the work area in the last 3 bytes of the parameter list pointed to by register 1 and described in Figure 20.

If you acquire the storage for the work area by using the GETMAIN macro instruction, you can request that it be freed by a FREEMAIN macro instruction after all information has been extracted from it. Set the high–order bit of the option byte in the work area to 1 and place the number of the subpool from which the work area was requested in byte 3 of the recovery work area.

Only one recovery attempt per data set is allowed during open, close, or end–of–volume processing. If a recovery attempt is unsuccessful, you may not request another recovery. The second time through the exit routine you may request only one of the other options (if allowed): issue the ABEND immediately, ignore the ABEND, or delay the ABEND. If at any time you select an option that is not allowed, the ABEND is issued immediately.

Note that if recovery is successful, you still receive an ABEND message on your listing. This message refers to the ABEND that would have been issued if the recovery had not been successful.

Modifying the Data Control Block

You can complete or modify the DCB during execution of your program. You can also determine data set characteristics from information supplied by the data set labels. Changes or additions can be made before opening of the data set, after closing it, during the DCB exit routine, or while the data set is open. Naturally, any information must be supplied before it is needed.

Because each DCB does not have a symbolic name for each field, a DCBD macro instruction must be used to supply the symbolic names. By loading a base register with the address of the DCB to be processed, you can refer to any field symbolically.

The DCBD macro instruction generates a dummy control section (DSECT) named IHADCB. The name of each field consists of DCB followed by the first five letters of the keyword operand that represents the field in the DCB macro instruction. For example, the field reserved for blocksize is referred to as DCBBLKSI.

The attributes of each DCB field are defined in the dummy control section. Because each field in the DCB is not necessarily aligned on a fullword boundary, care must be taken when storing or moving data into the field. The length attribute and the alignment of each field can be determined from an assembly listing of the DCBD macro instruction.

The DCBD macro instruction can be coded once to describe all DCBs even though their fields differ because of differences in data set organization and access technique. It must not be coded more than once for a single assembly. If it is coded before the end of a control section, it must be followed by a CSECT or DSECT statement to resume the original control section.

Changing an Address in the Data Control Block: Figure 23 illustrates how you can modify a field in the data control block. The DCBD macro instruction defines the symbolic name of each field.

The data set defined by the data control block TEXTDCB is opened for use as both an input and an output data set. When its use as an input data set is completed, the EODAD routine closes the data set temporarily to reposition the volume for output. The EODAD routine then uses the dummy control section IHADCB to change the error exit address (SYNAD) from INERROR to OUTERROR.

The EODAD routine loads the address TEXTDCB into register 10, which it uses as a base register for IHADCB. It then moves the address OUTERROR into the DCBSYNAD field of the DCB. This field is a fullword, but contains information that must not be disturbed in the high–order byte. For this reason, care must be taken to change only the 3 low–order bytes of the field.

```
            OPEN        (TEXTDCB,INOUT)
            ...
EOFEXIT     CLOSE       (TEXTDCB,REREAD),TYPE=T
            LA          10,TEXTDCB
            USING       IHADCB,10
            MVC         DCBSYNAD+1(3),=AL3(OUTERROR)
            B           OUTPUT
INERROR     STM         14,12,SYNADSA+12
            ...
OUTERROR    STM         14,12,SYNADSA+12
            ...
TEXTDCB     DCB         DSORG=PS,MACRF=(R,W),DDNAME=TEXTTAPE,          C
                      EODAD=EOFEXIT,SYNAD=INERROR
            DCBD        DSORG=PS
```

Figure 23. Modifying a Field in the Data Control Block

Sharing a Data Set

A data set can be shared by all the tasks of a job step. If requested in the DD statement, a data set can be shared by all the tasks in the system. However, regardless of which access method is used, the task that opens the data set must also close it.

When a data set is shared by several tasks, you must treat it as a serially reusable resource. A task must have exclusive control of a data set in order to add or update records, and shared control in order to read records.

In performing a task, you gain exclusive or shared control of a data set by issuing the ENQ and DEQ macro instructions, which are described in *OS Supervisor Services and Macro Instructions.* Note that these macro instructions must be used by all of the tasks that process a shared data set.

When you process a direct data set, you need to use the ENQ and DEQ macro instructions only when tasks that share a data set do not refer to the same DCB. When all tasks do refer to the same DCB, you must have exclusive control of a block of records that you are updating, but you do not need either shared or exclusive control of the entire data set. You can request exclusive control of a block of records through the DCB, READ, WRITE, and RELEX macro instructions.

Shared Direct–Access Storage Devices: At some installations, a direct–access storage device is shared by two or more independent computing systems. Tasks executed on these systems can share data sets stored on the device. For details, refer to *OS MFT Guide* or *OS MVT Guide.*

PART 2: DATA MANAGEMENT PROCESSING PROCEDURES

Data–Processing Techniques

The operating system allows you to concentrate most of your efforts on processing the records read or written by the data management routines. To get the records read and written, your main responsibilities are to describe the data set to be processed, the buffering techniques to be used, and the access method. An access method has been defined as the combination of data set organization and the technique used to gain access to the data. Data access techniques are discussed here in two categories — queued and basic.

Queued Access Technique

The queued access technique provides GET and PUT macro instructions for transmitting data between main and auxiliary storage. These macro instructions cause automatic blocking and deblocking of the records stored and retrieved. Anticipatory (look–ahead) buffering and synchronization (overlap) of input and output operations with central processing unit (CPU) processing are automatic features of the queued access technique.

Because the operating system controls buffer processing, you can use as many input/output (I/O) buffers as needed without reissuing GET or PUT macro instructions to fill or empty buffers. Usually, more than one input block is in main storage at any given time, so I/O operations do not delay record processing.

Because the operating system synchronizes input/output with processing, you need not test for completion, errors, or exceptional conditions. After a GET or PUT macro instruction is issued, control is not returned to your program until an input area is filled or an output area is available. Exits to error analysis (SYNAD) and end–of–volume or end–of–data (EODAD) routines are automatically taken when necessary.

GET — Retrieve a Record

The GET macro instruction obtains a record from an input data set. It operates in a logical sequential and device–independent manner. As required, the GET macro instruction schedules the filling of input buffers, deblocks records, and directs input error recovery procedures. For sequential data sets, it also merges record segments into logical records. After all records have been processed and the GET macro instruction detects an end–of–data indication, the system automatically checks labels on sequential data sets and passes control to your end–of–data (EODAD) routine. If an end–of–volume condition is detected for a sequential data set, the system provides automatic volume switching if the data set extends across several volumes or if concatenated data sets are being processed. If you specify OPTCD=Q in the DCB, GET causes input data to be translated from ASCII to EBCDIC.

PUT — Write a Record

The PUT macro instruction places a record into an output data set. Like the GET macro instruction, it operates in a logical sequential and device–independent manner. As required, the PUT macro instruction schedules the emptying of output buffers, blocks records, and handles output error correction procedures. For sequential data

sets, it also initiates automatic volume switching and label creation, and also segments records for spanning. If you specify OPTCD=Q in the DCB, PUT causes output to be translated from EBCDIC to ASCII.

If the PUT macro instruction is directed to a card punch or printer, the system automatically adjusts the number of records or record segments per block of format–F or format–V blocks to 1. Thus, you can specify a record length (LRECL) and blocksize (BLKSIZE) to provide an optimum blocksize if the records are temporarily placed on magnetic tape or a direct–access volume.

For spanned variable–length records, the blocksize must be equivalent to the length of one card or one print line. Record size may be greater than blocksize in this case.

PUTX — Write an Updated Record

The PUTX macro instruction is used to update a data set or to create an output data set using records from an input data set as a base. PUTX updates, replaces, or inserts records from existing data sets but does not create records or add records from other data sets.

When you use the PUTX macro instruction to update, each record is returned to the data set referred to by a previous GET macro instruction. The buffer containing the updated record is flagged and written back to the same location on the direct–access storage device from which it was read. The block is not written until a GET macro instruction is issued for the next buffer, except when a spanned record is to be updated. In that case, the block is written with the next GET macro instruction.

When the PUTX macro instruction is used to create an output data set, you can add new records by using the PUT macro instruction. As required, the PUTX macro instruction blocks records, schedules the writing of output buffers, and handles output error correction procedures.

Basic Access Technique

The basic access technique provides the READ and WRITE macro instructions for transmitting data between main and auxiliary storage. This technique is used when the operating system cannot predict the sequence in which the records are to be processed or when you do not want some or all of the automatic functions performed by the queued access technique. Although the system does not provide anticipatory buffering or synchronized scheduling, macro instructions are provided to help you program these operations.

The READ and WRITE macro instructions process blocks, not records. Thus, blocking and deblocking of records is your responsibility. Buffers, allocated by either you or the operating system, are filled or emptied individually each time a READ or WRITE macro instruction is issued. Moreover, the READ and WRITE macro instructions only initiate input/output operations. To ensure that the operation is completed successfully, you must issue a CHECK macro instruction to test the data event control block (DECB) or issue a WAIT macro instruction and then check the DECB yourself. The number of READ or WRITE macro instructions issued before a CHECK macro instruction is used should not exceed the specified number of channel programs (NCP).

READ — Read a Block

The READ macro instruction retrieves a data block from an input data set and places it in a designated area of main storage. To allow overlap of the input operation with processing, the system returns control to your program before the read operation is completed. The DECB created for the read operation must be tested for successful completion before the record is processed or the DECB is reused.

If an indexed sequential data set is being read, the block is brought into main storage and the address of the record is returned to you in the DECB.

When you use the READ macro instruction for BSAM to read a direct data set with spanned records and keys and you specify BFTEK=R in your DCB, the data management routines displace record segments after the first in a record by key length. Thus, you can expect the block descriptor word and the segment descriptor word at the same locations in your buffer or buffers, regardless of whether you read the first segment of a record, which is preceded in the buffer by its key, or a subsequent segment, which does not have a key. This procedure is called *offset reading.*

You can specify variations of the READ macro instruction according to the organization of the data set being processed and the type of processing to be done by the system as follows:

Sequential

- SF – Read the data set sequentially.
- SB – Read the data set backward (magnetic tape, format–F and format–U only). When RECFM=FBS, data sets containing a last truncated block cannot be read backward.

Indexed Sequential

- K – Read the data set.
- KU – Read for update. The system maintains the device address of the record; thus, when a WRITE macro instruction returns the record, no index search is required.

Direct

- D – Use the direct access method.
- I – Locate the block using a block identification.
- K – Locate the block using a key.
- F – Provide device position feedback.
- X – Maintain exclusive control of the block.
- R – Provide next address feedback.
- U – Next address can be a capacity record or logical record, whichever occurred first.

WRITE — Write a Block

The WRITE macro instruction places a data block in an output data set from a designated area of main storage. The WRITE macro instruction can also be used to return an updated record to a data set. To allow overlap of output operations with processing, the system returns control to your program before the write operation is completed. The DECB created for the write operation must be tested for successful completion before the DECB can be reused. For ASCII tape data sets, do not issue more than one WRITE on the same record, because the WRITE macro instruction causes the data in the record to be translated from EBCDIC to ASCII.

As with the READ macro instruction, you can specify variations of the WRITE macro instruction according to the organization of the data set and the type of processing to be done by the system as follows:

Sequential

- SF – Write the data set sequentially.
- SFR – Write the data set sequentially with next–address feedback.

Indexed Sequential

- K – Write a block containing an updated record, or replace a record with an unblocked record having the same key. The record to be replaced need not have been read into main storage.
- KN – Write a new record or change the length of a variable–length record.

Direct

- SD – Write a dummy fixed–length record.
- SZ – Write a capacity record (R0). The system supplies the data, writes the capacity record, and advances to the next track.
- D – Use the direct access method.
- I – Search argument identifies a block.
- K – Search argument is a key.
- A – Add a new block.
- F – Provide record location data (feedback).
- X – Release exclusive control.

CHECK — Test Completion of Read or Write Operation

When processing a data set, you can test for completion of a READ or WRITE request by issuing a CHECK macro instruction. The system tests for errors and exceptional conditions in the data event control block (DECB). Successive CHECK macro instructions issued for the same data set must be issued in the same order as the associated READ and WRITE macro instructions.

The check routine passes control to the appropriate exit routines specified in the DCB for error analysis (SYNAD) or, for sequential data sets, end–of–data (EODAD). It also automatically initiates end–of–volume procedures (volume switching or extending output data sets).

If you specify OPTCD=Q in the DCB, CHECK causes input data to be translated from ASCII to EBCDIC.

WAIT — Wait for Completion of a Read or Write Operation

When processing a data set, you can test for completion of any READ or WRITE request by issuing a WAIT macro instruction. The input/output operation is synchronized with processing, but the DECB is not checked for errors or exceptional conditions, nor are end–of–volume procedures initiated. Your program must perform these operations.

The WAIT macro instruction can be used to await completion of multiple read and write operations. Each operation must then be checked or tested separately.

Data Event Control Block (DECB)

A data event control block is a 16– to 32–byte area reserved by each READ or WRITE macro instruction. It contains control information and pointers to standard

status indicators. It is described in detail in Appendix A of *OS Data Management Macro Instructions.*

The DECB is examined by the Check routine when the I/O operation is completed to determine if an uncorrectable error or exceptional condition exists. If it does, control is passed to your SYNAD routine. If you have no SYNAD routine, the task is abnormally terminated.

Error Handling

The basic and queued access techniques both provide special macro instructions for analyzing input/output errors. These macro instructions can be used in SYNAD routines and in error analysis routines that are entered directly when you use the basic access technique with indexed sequential data sets.

SYNADAF — Perform SYNAD Analysis Function

The SYNADAF macro instruction analyzes the status, sense, and exceptional condition code data that is available to your error analysis routine. It produces an error message that your routine can write into any appropriate data set. The message is in the form of an unblocked variable–length record, but you can write it as a fixed–length record by omitting the block length and record length fields that precede the message text.

The text of the message is 120 characters long, and begins with a field of 36 or 42 blanks; you can use the blank field to add your own remarks to the message. Following is a typical message with the blank field omitted:

```
,TESTJOB ,STEP2 ,283,TA,MASTER ,READ ,DATA CHECK ,0000015,BSAM
```

This message indicates that a data check occurred during reading of the fifteenth block of a data set. The data set was identified by a DD statement named MASTER, and was on a magnetic-tape volume on unit 283. The name of the job was TESTJOB; the name of the job step was STEP2.

If the error analysis routine is entered because of an input error, the first 6 bytes of the message (bytes 8–13) contain binary information. If no data was transmitted or if the access method is QISAM, the first 6 bytes are blank. If the error did not prevent data transmission, the first 6 bytes contain the address of the input buffer and the number of bytes read. You can use this information to process records from the block; for example, you might print each record after printing the error message. Before printing the message, however, you should replace this binary information with EBCDIC characters.

The SYNADAF macro instruction provides its own save area and makes this area available to your error analysis routine. When used at the entry point of a SYNAD routine, it fulfills the routine's responsibility for providing a save area.

SYNADRLS — Release SYNADAF Message and Save Areas

The SYNADRLS macro instruction releases the message and save areas provided by the SYNADAF macro instruction. You must issue this macro instruction before returning from the error analysis routine.

ATLAS — Perform Alternate Track Location Assignment

The ATLAS macro instruction enables your program to recover from permanent input/output errors when processing a data set in direct–access storage. After a data

check, or in certain missing–address–marker conditions, you can issue ATLAS to assign an alternate track to replace the error track or transfer data from the error track to the alternate track.

Use of the ATLAS macro instruction requires a knowledge of channel programming. A detailed description of the macro instruction and its use is included in *OS Data Management for System Programmers.*

If you do not use the ATLAS macro instruction, you can use the IEHATLAS utility program to perform the same function. The principal difference between the macro instruction and the utility program is that the latter provides error recovery only after your own program has been completed. For a detailed description of IEHATLAS, refer to *OS Utilities.*

Selecting an Access Method

Access methods are identified primarily by the data set organization to which they apply. For instance, BDAM is the basic access method for direct organization. Nevertheless, there are times when an access method identified with one organization can be used to process a data set usually thought of as organized in a different manner. Thus, a data set created by the basic access method for sequential organization (BSAM) may be processed by the basic direct access method (BDAM). If the queued access technique is used to process a sequential data set, the access method is referred to as the queued sequential access method (QSAM).

Basic access methods are used for all data organizations, while queued access methods apply only to sequential and indexed sequential data sets as shown in Figure 24.

Data Set Organization	Access Technique	
	Basic	**Queued**
Sequential	BSAM	QSAM
Partitioned	BPAM	
Indexed Sequential	BISAM	QISAM
Direct	BDAM	

Figure 24. Data Management Access Methods

It is possible to control an I/O device directly while processing a data set with any data organization without using a specific access method. The execute channel program (EXCP) macro instruction uses the system programs that provide for scheduling and queuing I/O requests, efficient use of channels and devices, data protection, interruption procedures, error recognition and retry. Complete details about the EXCP macro are in *OS Data Management for System Programmers.*

Opening and Closing a Data Set

Although your program has been assembled, the various data management routines required for I/O operations are not a part of the object code. In other words, your program is not completely assembled until the DCBs are initialized for execution. You accomplish initialization by issuing the OPEN macro instruction. After all DCBs have been completed, the system ensures that all required access method routines are loaded and ready for use and that all channel command word lists and buffer areas are ready.

Access method routines are selected and loaded according to data control fields that indicate:

- Data organization
- Buffering technique
- Access technique
- I/O unit characteristics

This information is used by the system to allocate main–storage space and load the appropriate routines. These routines, the channel command word (CCW) lists, and buffer areas created automatically by the system remain in main storage until the Close routine signals that they are no longer needed by the DCB that was using them.

When I/O operations for a data set are completed, you should issue a CLOSE macro instruction to return the DCB to its original status, handle volume disposition, create data set labels, complete writing of queued output buffers, and free main and auxiliary storage.

After closing the data set, you should issue a FREEPOOL macro instruction to release the main storage used for the buffer pool. If you plan to process other data sets, use FREEPOOL to regain the buffer pool storage space. If you expect to reopen a data set using the same DCB, use FREEPOOL unless the buffer pool created the first time the data set was opened will meet your needs when you reopen the data set. FREEPOOL is discussed in more detail in the section "Buffer Pool Construction."

After the data set has been closed, the DCB can be used for another data set. If you do not close the data set before a task terminates, the operating system closes it automatically. If the DCB is not available to the system at that time, the operating system abnormally terminates the task, and data results can be unpredictable. Note, however, that the operating system cannot automatically close any open data sets after the normal termination of a program that was brought into main storage by the loader. Therefore, loaded programs must include CLOSE macro instructions for all open data sets.

An OPEN or CLOSE macro instruction can be used to initiate or terminate processing of more than one data set. Simultaneous opening or closing is faster than issuing separate macro instructions; however, additional storage space is required for each data set specified.

Notes:

- Two or more DCBs should never be opened concurrently for output to the same data set on a direct–access device, except with the indexed sequential access method (ISAM). Otherwise the end–of–file record written by the CLOSE for one DCB may overlay data associated with another DCB.
- Two or more DCBs should never be opened concurrently using the same DDname. This is true for both input and output and especially important when you are using more than one access method. Any action on one DCB that alters the task input/output table (TIOT) or JFCB affects the other DCB(s) and thus can cause unpredictable results.
- If you want to use the same DD statement for two or more DCBs, you cannot specify parameters for fields in the first DCB and then be assured of obtaining the default parameters for the same fields in any subsequent DCB using the same

DD statement. Therefore, unless the parameters of all DCBs using one DD statement are the same, you should use separate DD statements.

- Associated data sets for the 3525 Card Punch can be opened in any order, but all data sets must be opened before any processing can begin. Associated data sets can be closed in any order, but once a data set has been closed, I/O operations cannot be performed on any of the associated data sets. See "Appendix C: Special Programming Considerations for the 3505 Card Reader and the 3525 Card Punch" for more information.

Volume disposition specified in the OPEN or CLOSE macro instruction can be overridden by the system if necessary. However, you need not be concerned; the system automatically requests the mounting and demounting of volumes, depending upon the availability of devices at a particular time.

OPEN — Initiate Processing of a Data Set

The OPEN macro instruction is used to complete a data control block for an associated data set. The method of processing and the volume positioning instruction in the event of an end–of–volume condition can be specified.

Processing Method: You can process a data set as either input or output (by coding INPUT or OUTPUT as the processing method operand of the OPEN macro) or, under BSAM, a combination of the two (by coding INOUT or OUTIN). Associated data sets to be processed concurrently on the 3525 Card Punch must have separate DCBs for each data set. See "Appendix C: Special Programming Considerations for the 3505 Card Reader and the 3525 Card Punch" for more information. If the data set resides on a direct–access volume, you can code UPDAT in the processing method operand to indicate that records can be updated. To specify that a magnetic–tape volume is to be read backward by BSAM or QSAM, code RDBACK in this operand. If the processing method operand is omitted from the OPEN macro instruction, INPUT is assumed. The operand is ignored by the basic indexed sequential access method (BISAM); it must be specified as OUTPUT when you are using the queued indexed sequential access method (QISAM) to create an indexed sequential data set. You can override the INOUT and OUTIN at execution by using the LABEL parameter of the DD statement, as discussed in *OS Job Control Language Reference.*

Simultaneous Opening of Data Sets: In Figure 25, the data sets associated with three DCBs are to be opened simultaneously.

```
         OPEN    (TEXTDCB,,CONVDCB,(OUTPUT),PRINTDCB,(OUTPUT))
+        CNOP    0,4
+        BAL     1,*+16          Load register with list address
+        DC      AL1(0)          Option byte
+        DC      AL3(TEXTDCB)    DCB address
+        DC      AL1(15)         Option byte
+        DC      AL3(CONVDCB)    DCB address
+        DC      AL1(143)        Option byte
+        DC      AL3(PRINTDCB)   DCB address
+        SVC     19              Issue open SVC
```

Figure 25. Opening Three Data Sets Simultaneously

Since no processing method operand is specified for TEXTDCB, the system assumes INPUT. Both CONVDCB and PRINTDCB are opened for output. No volume

positioning options are specified; thus, the position indicated by the DD statement DISP parameter is used.

At execution, the SVC 19 instruction passes control to the Open routine, which then initializes the three DCBs and loads the appropriate access method routines.

CLOSE — Terminate Processing of a Data Set

The CLOSE macro instruction is used to terminate processing of a data set and release it from a DCB. The volume positioning that is to result from closing the data set can also be specified. Volume positioning options are the same as those that can be specified for end–of–volume conditions in the OPEN macro instruction or the DD statement. An additional volume positioning option, REWIND, is available and can be specified by the CLOSE macro instruction for magnetic–tape volumes. REWIND positions the tape at the load point regardless of the direction of processing.

The operating system provides a temporary closing option, CLOSE (TYPE=T), for data sets being processed by BSAM. This option cannot be used if you are using BSAM to create a direct data set (MACRF=WL coded in the DCB macro instruction). CLOSE (TYPE=T) causes the RLSE parameter on the DD card to be ignored. When the macro instruction is executed for data sets on magnetic–tape or direct–access volumes, the system processes labels and repositions the volume as required. However, the DCB maintains its open status. You can continue processing of the data set at a later stage in your program without reissuing the OPEN macro instruction, thus improving performance. Magnetic–tape volumes are repositioned to the point either preceding the first data block or following the last data block of the data set. The presence of tape labels has no effect on repositioning.

Simultaneous Closing of Data Sets: In Figure 26, the data sets associated with three DCBs are to be closed simultaneously.

```
         CLOSE    (TEXTDCB,,CONVDCB,,PRINTDCB)
+        CNOP     0,4
+        BAL      1,*+16                  Branch around list
+        DC       AL1(0)                  Option byte
+        DC       AL3(TEXTDCB)            DCB address
+        DC       AL1(0)                  Option byte
+        DC       AL3(CONVDCB)            DCB address
+        DC       AL1(128)                Option byte
+        DC       AL3(PRINTDCB)           DCB address
+        SVC      20                      Issue close SVC
```

Figure 26. Closing Three Data Sets Simultaneously

Because no volume positioning operands are specified, the position indicated by the DD statement DISP parameter is used.

At execution, the SVC 20 instruction passes control to the Close routine, which terminates processing of the three data sets and returns the three DCBs to their original status.

End–of–Volume Processing

Control is passed automatically to the data management end–of–volume routine when any of the following conditions is detected:

- End–of–data indicator (input volume)
- Tapemark (input tape volume)
- Filemark (input direct–access volume)
- End of reel (output tape volume)
- End of extent (output direct–access volume)

You may issue a force end–of–volume (FEOV) macro instruction before the end–of–volume condition is detected.

The end–of–volume routine checks or creates standard trailer labels, if the LABEL parameter of the associated DD statement indicates standard labels. Control is then passed to the appropriate user label routine if it is specified in your exit list.

If multiple–volume data sets are specified in your DD statement, automatic volume switching is accomplished by the end–of–volume routine. When an end–of–volume condition exists on an output data set, additional space is allocated as indicated in your DD statement. If no more volumes are specified or if more than specified are required, the storage is obtained from any available volume on a device of the same type. If no such volume is available, your job is terminated.

Volume Positioning: When an end–of–volume condition is detected, the system positions the volume according to the disposition specified in the DD statement unless the volume disposition is specified in the OPEN macro instruction. Volume positioning instructions for a sequential data set on tape or direct access can be specified as LEAVE or REREAD.

LEAVE

positions the volume at the logical end of the data set just read or written. If the data set has been read backward, the logical end is the physical beginning of the data set.

REREAD

positions the volume at the logical beginning of the data set just read or written.

A volume positioning instruction can be specified only if the processing method operand has been specified. It is ignored if devices other than magnetic–tape and direct–access are used, or if the number of volumes exceeds the number of available units.

For magnetic–tape volumes, positioning varies according to the direction of the last input operation and the existence of tape labels. If the tape was last read forward:

LEAVE

positions a labeled tape to the point following the tapemark that follows the data set trailer label group, and an unlabeled volume to the point following the tapemark that follows the last block of the data set.

REREAD

positions a labeled tape to the point preceding the data set header label group, and an unlabeled tape to the point preceding the first block of the data set.

If the tape was last read backward:

LEAVE

positions a labeled tape to the point preceding the data set header label group, and an unlabeled tape to the point preceding the first block of the data set.

REREAD

positions a labeled tape to the point following the tapemark that follows the data set trailer label group, and an unlabeled tape to the point following the tapemark that follows the last block of the data set.

FEOV — Force End of Volume

The FEOV macro instruction directs the operating system to initiate end–of–volume processing before the physical end of the current volume is reached. If another volume has been specified for the data set, volume switching takes place automatically. The volume positioning options REWIND and LEAVE are available.

The FEOV macro instruction can only be used when you are using BSAM or QSAM.

Buffer Acquisition and Control

The operating system provides several methods of buffer acquisition and control. Each *buffer* (main–storage area used for intermediate storage of input/output data) usually corresponds in length to the size of a block in the data set being processed. When you use the queued access technique, any reference to a buffer actually refers to the next record *(buffer segment).*

You can assign more than one buffer to a data set by associating the buffer with a *buffer pool.* A buffer pool must be constructed in a main–storage area allocated for a given number of buffers of a given length.

Buffer segments and buffers within the buffer pool are controlled automatically by the system when the queued access technique is used. However, you can terminate processing of a buffer by issuing a release (RELSE) macro instruction for input or a truncate (TRUNC) macro instruction for output. Two buffering techniques, simple and exchange, can be used to process a sequential data set. Only simple buffering can be used to process an indexed sequential data set.

If you use the basic access technique, you can use buffers as work areas rather than as intermediate storage areas. You can control them directly, by using the GETBUF and FREEBUF macro instructions, or dynamically, by requesting dynamic buffering in your DCB macro instruction and your READ or WRITE macro instruction. If you request dynamic buffering, the system will automatically provide a buffer each time a READ macro instruction is issued. That buffer will be freed when you issue a WRITE or FREEDBUF macro instruction.

Buffer Pool Construction

Buffer pool construction can be accomplished in any of three ways:

- Statically using the BUILD macro instruction
- Explicitly using the GETPOOL macro instruction
- Automatically by the system when the data set is opened

If QSAM simple buffering is used, the buffers are automatically returned to the pool when the data set is closed. If the buffer pool is constructed explicitly or automatically, the main storage area must be returned to the system by the FREEPOOL macro instruction.

In many applications, fullword or doubleword alignment of a block within a buffer is important. You can specify in the DCB that buffers are to start on either a doubleword boundary or a fullword boundary that is not also a doubleword boundary (by coding BFALN=D or F). If doubleword alignment is specified for format–V records, the fifth byte of the first record in the block is so aligned. For that reason, fullword alignment must be requested to align the first byte of the variable–length record on a doubleword boundary. The alignment of the records following the first in the block depends on the length of the previous records.

Note that buffer alignment provides alignment only for the buffer. If records from ASCII magnetic tape are read and the records use the block prefix, the boundary alignment of logical records within the buffer depends on the length of the block prefix. If the length is 4, logical records are on fullword boundaries. If the length is 8, logical records are on doubleword boundaries.

If the BUILD macro instruction is used to construct the buffer pool, alignment depends on the alignment of the first byte of the reserved storage area.

When you process multiple QISAM data sets, you can use a common buffer pool. To do this, however, you must use the BUILD macro instruction to reformat the buffer pool before opening each data set.

BUILD — Construct a Buffer Pool

When you know, before program assembly, both the number and the size of the buffers required for a given data set, you can reserve an area of appropriate size to be used as a buffer pool. Any type of area can be used — for example, a predefined storage area or an area of coding no longer needed.

A BUILD macro instruction, issued during execution of your program, structures the reserved storage area into a buffer pool. The address of the buffer pool must be the same as that specified for the buffer pool control block (BUFCB) in your DCB. The buffer pool control block is an 8–byte field preceding the buffers in the buffer pool. The number (BUFNO) and length (BUFL) of the buffers must also be specified. For QSAM, the length of BUFL must be at least the blocksize.

When the data set using the buffer pool is closed, you can reuse the area as required. You can also reissue the BUILD macro instruction to reconstruct the area into a new buffer pool to be used by another data set.

You can assign the buffer pool to two or more data sets that require buffers of the same length. To do this, you must construct an area large enough to accommodate the total number of buffers required at any one time during execution. That is, if each of two data sets requires five buffers (BUFNO=5), the BUILD macro instruction should specify ten buffers. The area must also be large enough to contain the 8–byte buffer pool control block.

BUILDRCD — Build a Buffer Pool and a Record Area

The BUILDRCD macro instruction performs the same functions as the BUILD macro instruction and the following additional functions:

- It allows you access to an entire logical record, not just a segment, for a sequential data set used by QSAM in locate mode and having a record format of VS or VBS.
- It links a record area to the buffer control block by extending the buffer control block to 12 bytes. Thus, a spanned record can be assembled or segmented in the record area.

GETPOOL — Get a Buffer Pool

If a specified area is not reserved for use as a buffer pool, or you want to defer specifying the number and length of the buffers until execution of your program, you should use the GETPOOL macro instruction. It enables you to vary the size and number of buffers according to the needs of the data set being processed.

The GETPOOL macro instruction structures a main–storage area allocated by the system into a buffer pool, assigns a buffer pool control block, and associates the pool with a specific data set. The GETPOOL macro instruction should be issued either before opening of the data set or during your DCB exit routine.

When using GETPOOL with QSAM, specify a buffer length (BUFL) of at least the blocksize.

Automatic Buffer Pool Construction

If you have requested a buffer pool and have not used an appropriate macro instruction by the end of your DCB exit routine, the system automatically allocates main–storage space for a buffer pool. The buffer pool control block is also assigned and the pool is associated with a specific DCB. If you are using the basic access technique to process an indexed sequential or direct data set, you must indicate dynamic buffer control. Otherwise, the system does not construct the buffer pool automatically.

Because a buffer pool obtained automatically is not freed automatically when you issue a CLOSE macro instruction, you should also issue a FREEPOOL macro instruction, which is discussed in the next section.

FREEPOOL — Free a Buffer Pool

Any buffer pool assigned to a DCB either automatically by the OPEN macro instruction (except when dynamic buffer control is used) or explicitly by the GETPOOL macro instruction must be released before your program is terminated. The FREEPOOL macro instruction should be issued to release the main storage area as soon as the buffers are no longer needed. As a general rule, when you are using the queued access technique, an output data set should be closed first to ensure that all the records have been written. However, when you are using exchange buffering or when

processing an indexed sequential data set using the queued access technique, the buffer pool must not be released until all the data sets have been closed.

Constructing a Buffer Pool: Figures 27 and 28 illustrate several possible methods of constructing a buffer pool. They do not take into account the method of processing or controlling the buffers in the pool.

```
          ...                                          Processing
          BUILD     (INDCB,,OUTDCB,(OUTPUT))           Structure a buffer pool
          ...                                          Processing
          ...                                          Processing
ENDJOB    CLOSE     (INDCB,,OUTDCB)
                ...                                    Processing
          RETURN                                       Return to system control
INDCB     DCB       BUFNO=5,BUFCB=INPOOL,EODAD=ENDJOB,---
OUTDCB    DCB       BUFNO=5,BUFCB=INPOOL,---
          CNOP      0,8                                Force boundary alignment
INPOOL    DS        CL528                              Buffer pool
```

Figure 27. Constructing a Buffer Pool From a Static Storage Area

In Figure 27, a static storage area named INPOOL is allocated during program assembly. The BUILD macro instruction, issued during execution, arranges the buffer pool into ten buffers, each 52 bytes long. Five buffers are assigned to INDCB and five to OUTDCB, as specified in the DCB macro instruction for each. The two data sets share the buffer pool because both specify INPOOL as the buffer pool control block. Notice that an additional 8 bytes have been allocated for the buffer pool to contain the buffer pool control block.

```
          ...
          GETPOOL   INDCB,10,52                        Construct a 10-buffer pool
          GETPOOL   OUTDCB,5,112                       Construct a 5-buffer pool
          OPEN      (INDCB,,OUTDCB,(OUTPUT))
          ...
ENDJOB    CLOSE     (INDCB,,OUTDCB)
          FREEPOOL  INDCB                              Release buffer pools after all
                                                       I/O is complete
          FREEPOOL  OUTDCB
          ...
          RETURN                                       Return to system control
INDCB     DCB       DSORG=PS,BFALN=F,LRECL=52,RECFM=F,EODAD=ENDJOB,---
OUTDCB    DCB       DSORG=IS,BFALN=D,LRECL=52,KEYLEN=10,BLKSIZE=104,          C
                RKP=0,RECFM=FB,---
```

Figure 28. Constructing a Buffer Pool Using GETPOOL and FREEPOOL

In Figure 28, two buffer pools are constructed explicitly by the GETPOOL macro instructions. Ten input buffers are provided, each 52 bytes long, to contain one fixed–length record; five output buffers are provided, each 112 bytes long, to contain two blocked records plus an 8–byte count field (required by ISAM). Notice that both data sets are closed before the buffer pools are released by the FREEPOOL macro instructions. The same procedure should be used if the buffer pools were constructed automatically by the OPEN macro instruction.

Buffer Control

Your program can use four techniques to control the buffers used by your program. The advantages of each depend to a great extent upon the type of job you are doing. Simple and exchange buffering are provided for the queued access technique. The basic access technique provides for either direct or dynamic buffer control.

Although only simple buffering can be used to process an indexed sequential data set, buffer segments and buffers within a buffer pool are controlled automatically by the operating system.

In addition, the queued access technique provides four processing modes that determine the extent of data movement in main storage. Move, data, locate, or substitute mode processing can be specified for either the GET or PUT macro instruction. The buffer processing mode is specified in the MACRF field of the DCB macro instruction. The movement of a record is determined as follows:

- *Move mode:* The record is moved from an input buffer to your work area, or from your work area to an output buffer.
- *Data mode (QSAM format–V spanned records only):* The same as the move mode except only the data portion of the record is moved.
- *Locate mode:* The record is not moved. Instead, the address of the next input or output buffer is placed in register 1. For QSAM format–V spanned records, if you have specified logical records by specifying BFTEK=A or by issuing the BUILDRCD macro instruction, the address returned in register 1 points to a record area where the spanned record is assembled or segmented.
- *Substitute mode:* The record is not moved. Instead, the address of the next input or output buffer is interchanged with the address of your work area.

Two processing modes of the PUTX macro instruction can be used in conjunction with a GET–locate macro instruction. The update mode returns an updated record to the data set from which it was read; the output mode transfers an updated record to an output data set. There is no actual movement of data in main storage. The processing mode is specified by the operand of the PUTX macro instruction, as explained in *OS Data Management Macro Instructions.*

If you use the basic access technique, you can control buffers in one of two ways:

- Directly, using the GETBUF macro instruction to retrieve a buffer constructed as described above. A buffer can then be returned to the pool by the FREEBUF macro instruction.
- Dynamically, by requesting a dynamic buffer in your READ or WRITE macro instruction. This technique can be used only when you are processing an indexed sequential or direct data set. If you request dynamic buffering, the system automatically provides a buffer each time a READ macro instruction is issued. The buffer is supplied from a buffer pool that is created by the system when the data set is opened. The buffer is released (returned to the pool) upon completion of a WRITE macro instruction when you are updating. If you do not update the record in the buffer and thus release the buffer when the record is written, the FREEDBUF macro instruction may be used. If you are processing an indexed sequential data set, the buffer is automatically released upon the next issuance of the READ macro instruction if there has been no intervening WRITE or FREEDBUF macro instruction.

Simple Buffering

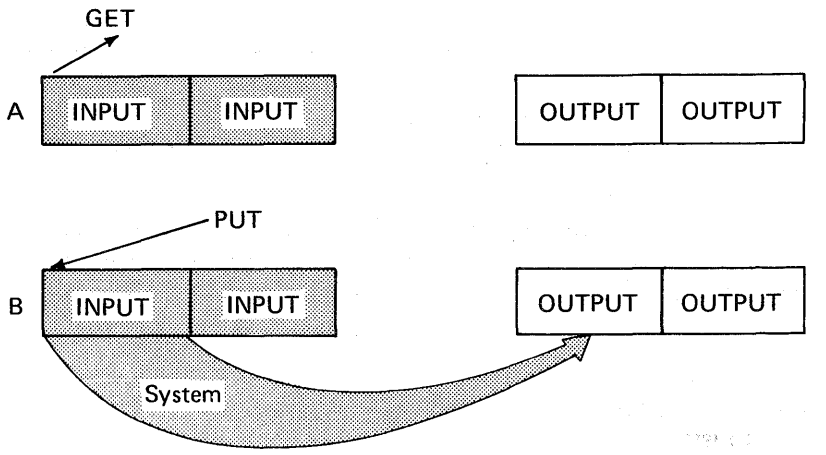
The term *simple buffering* refers to the relationship of segments within the buffer. All segments in a simple buffer are together in main storage and are always associated with the same data set. When the buffer pool is constructed, the system creates a channel command word (CCW) for each buffer in the buffer pool. For this reason, each record must be physically moved from an input buffer segment to an output buffer segment. It can be processed within either segment or in a work area.

If you use simple buffering, records of any format can be processed. New records can be inserted and old records deleted as required to create a new data set. A record can be moved and processed as follows:

- Processed in an input buffer and then moved to an output buffer (GET–locate, PUT–move/PUTX–output)
- Moved from an input buffer to an output buffer where it can be processed (GET–move, PUT–locate)
- Moved from an input buffer to a work area where it can be processed and then moved to an output buffer (GET–move, PUT–move)
- Processed in an input buffer and returned to the data set (GET–locate, PUTX–update)

The following examples illustrate the control of simple buffers and the processing modes that can be used. The buffer pools may have been constructed in any way previously described.

Simple Buffering — GET–locate, PUT–move/PUTX–output: The GET macro instruction (step A, Figure 29) locates the next input record to be processed. Its address is returned in register 1 by the system. The address is passed to the PUT macro instruction in register 0.




```
NEXTREC  GET     INDCB
         ...
         LR      0,1
         PUT     OUTDCB,(0)
         B       NEXTREC
INDCB    DCB     MACRF=(GL),---
OUTDCB   DCB     MACRF=(PM),---
```

Figure 29. Simple Buffering with MACRF=GL and MACRF=PM

The PUT macro instruction (step B, Figure 29) specifies the address of the record in register 0. The system then moves the record to the next output buffer.

Note: The PUTX–output macro instruction can be used in place of the PUT–move macro instruction. However, processing will be as described under exchange buffering (see PUT–substitute).

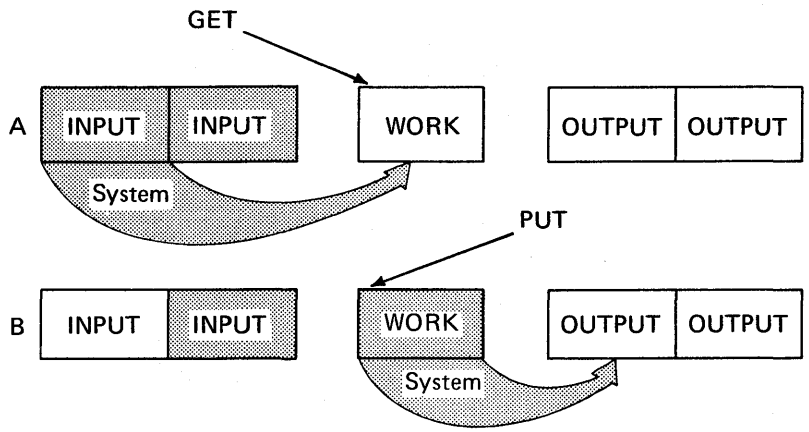
Simple Buffering — GET–move, PUT–locate: The PUT macro instruction locates the address of the next available output buffer. Its address is returned in register 1 and is passed to the GET macro instruction in register 0.

The GET macro instruction specifies the address of the output buffer into which the system moves the next input record.

A filled output buffer is not written until the next PUT macro instruction is issued.

Simple Buffering — GET–move, PUT–move: The GET macro instruction (step A, Figure 30) specifies the address of a work area into which the system moves the next record from the input buffer.

The PUT macro instruction (step B, Figure 30) specifies the address of a work area from which the system moves the record into the next output buffer.




```
NEXTREC   GET   INDCB,WORKAREA
          ...
          PUT   OUTDCB,WORKAREA
          B     NEXTREC
WORKAREA  DS    CL50
INDCB     DCB   MACRF=(GM),---
OUTDCB    DCB   MACRF=(PM),---
```

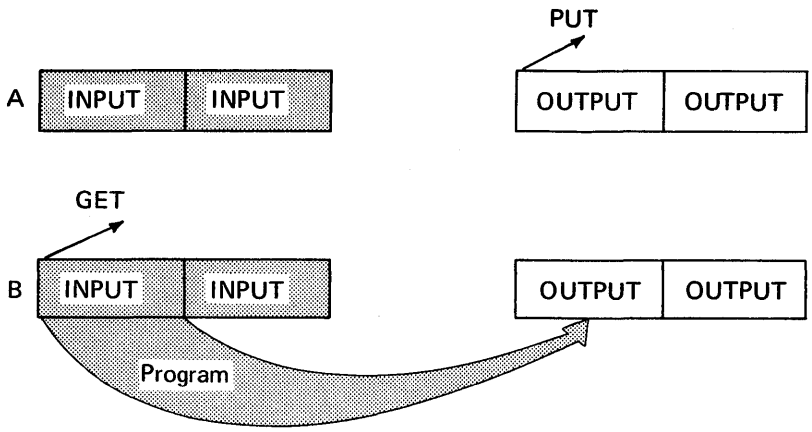
Figure 30. Simple Buffering with MACRF=GM and MACRF=PM

Simple Buffering — GET–locate, PUT–locate: The PUT macro instruction (step A, Figure 31) locates the address of the next available output buffer. The address is returned in register 1.

The GET macro instruction (step B, Figure 31) locates the address of the next input buffer. Its address is returned in register 1. You must then move the record from the input buffer to the output buffer. Processing can be done either before or after the move operation.

A filled output buffer is not written until the next PUT macro instruction is issued. Note that the last CLOSE macro instruction attempts to write the last record of your data set. Be careful not to issue an extra PUT before issuing CLOSE. Otherwise, when the CLOSE macro instruction tries to write your last record, it will write a meaningless record.

Note that if records other than format–F records are being moved, the length attribute of the MVC instruction must be changed as shown by the code beginning with the USING statement in Figure 31. If the record is more than 256 bytes, you will have to code a move routine to process the complete record.




```
NEXTREC  PUT    OUTDCB
         LR     6,1
         GET    INDCB
         LR     7,1
         USING  IHADCB,5
         LA     5,INDCB
         LH     4,DCBLRECL
         SH     4,=H'1'
         EX     4,MOVEREC
         ...
         B      NEXTREC
MOVEREC  MVC    0(1,6),0(7)
INDCB    DCB    MACRF=(GL),
                EODAD=EOF,---
OUTDCB   DCB    MACRF=(PL),---
         DCBD   DSORG=(LR)
EOF      ...
```

Figure 31. Simple Buffering with MACRF=GL and MACRF=PL

Exchange Buffering

The term *exchange buffering* refers to the relationship of segments within a buffer. All the segments in an exchange buffer are not necessarily together in main storage, nor are they always associated with the same data set. When the buffer pool is constructed, the system creates a channel command word (CCW) for each buffer segment in the buffer. This makes it possible to exchange the CCWs of different storage locations.

To use exchange buffering, you must provide a work area comparable in size and alignment to a buffer segment. That work area is substituted for the next buffer segment (the storage areas change roles). The CCW created for the buffer segment actually points to the work area.

Why use exchange buffering? Because there is no need to move the record. This means a considerable saving in processing time when you use substitute mode or PUTX–output mode.

The use of exchange buffering during execution of your program requires these conditions:

- Input and output buffers must be of the same size and alignment.
- Records must be unblocked or blocked format–F records.
- ASCII records must be format–F records with BUFOFF=(0).
- Track overflow cannot be used with blocked format–F records.
- GET–move and PUT–locate modes cannot be used.
- Unit–record devices must not be specified.

If you request exchange buffering, but it cannot be used, the system automatically uses simple buffering. Move mode processing is used in place of substitute mode.

After opening the data set, you can test the DCBCIND1 field of the DCB to determine if simple buffering was substituted for exchange buffering because of inconsistencies in the DCB information. The eighth bit of the DCBCIND1 field is 1 for exchange buffering and 0 for simple buffering.

If your records are blocked format–F records, each segment is aligned as specified in the DCBBFALN field. Therefore, your buffer length (DCBBUFL) must specify buffers large enough to contain segments whose length is a multiple of 16 bytes. Otherwise, the specified boundary alignment cannot be achieved; simple buffering is used and only the first byte in the first record is aligned as specified.

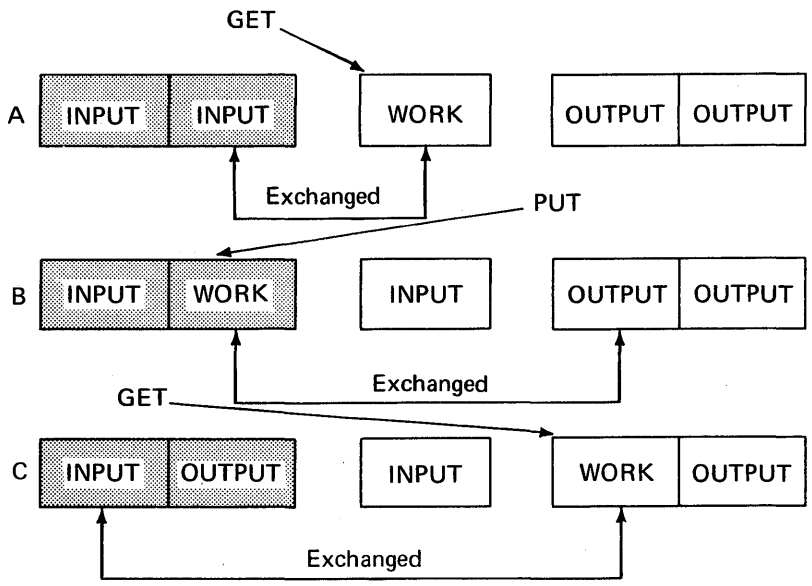
To reopen a DCB that has been opened for exchange buffering, you must first close all DCBs using the buffer pool associated with the DCB to be reopened and issue a FREEPOOL macro instruction specifying the DCB to be reopened.

There are two possible error conditions that cannot be prechecked by the system:

- Word alignment that does not correspond to the characteristics of the machine. If, for example, you expect to process your data on a System/360 Model 65, 75, 85, or 91 or on a System/370, your record length should be a multiple of 16; on a model 50, a multiple of 8; on a model 40, a multiple of 4. No error will result if the records are processed on a smaller system.
- An I/O device that transfers the data faster than the CPU can exchange the addresses in the CCW.

The following examples illustrate the control of exchange buffers and the corresponding processing modes that can be used. The buffer pools may have been constructed in any way previously described.

Exchange Buffering — GET–substitute, PUT–substitute: The GET macro instruction (step A, Figure 32) specifies the address of a work area. The work area address is exchanged with the address of the next input record returned in register 1. After processing, the address of the record is passed to the PUT macro instruction.




```
            LA      0,WORKAREA
NEXTREC     GET     INDCB,(0)
            LR      0,1
            ...
            PUT     OUTDCB,(0)
            LR      0,1
            B       NEXTREC
WORKAREA    DS      CL50
INDCB       DCB     MACRF=(GT),---
OUTDCB      DCB     MACRF=(PT),---
```

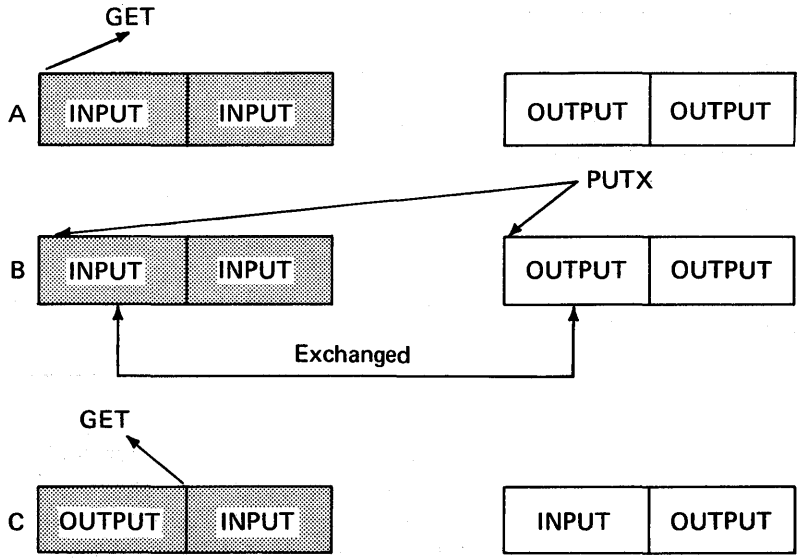
Figure 32. Exchange Buffering with MACRF=GT and MACRF=PT

The PUT macro instruction (step B, Figure 32) specifies the address of the output record. The output record address is exchanged for the address of the next output buffer available for use as a work area. The work area address, returned in register 1, is passed to the GET macro instruction (step C, Figure 32) in register 0.

Notice that as the areas are exchanged there is no movement of data. Output records are contained in the original input area and vice versa, but are logically associated with the correct data set.

Exchange Buffering — GET–locate, PUTX–output: The GET macro instruction (step A, Figure 33) locates the address of the next input record. The address is returned in register 1. The record must be processed in the buffer segment before the PUTX macro instruction (step B, Figure 33) is issued. The PUTX macro instruction specifies the address of both the input and output data control block. The two buffer segments are exchanged without any movement of data. The GET macro instruction (step C, Figure 33) locates the next record to be processed.

Notice that the DCB macro instruction for the output data set specifies move mode; this is required.




```
NEXTREC  GET     INDCB
         ...
         PUTX    OUTDCB,INDCB
         B       NEXTREC
INDCB    DCB     MACRF=(GL),---
OUTDCB   DCB     MACRF=(PM),---
```

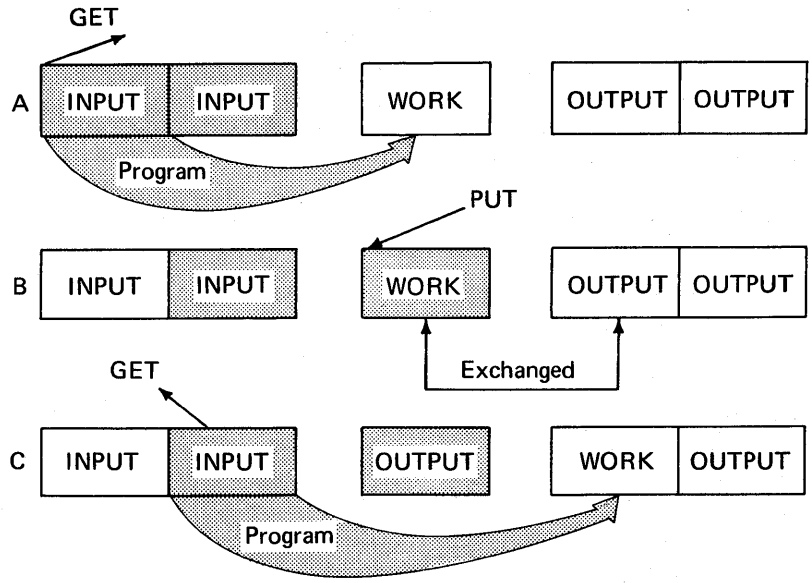
Figure 33. Exchange Buffering with MACRF=GL and MACRF=PM

Exchange Buffering — GET–locate, PUT–substitute: The GET macro instruction (step A, Figure 34) locates the next input record. Its address is returned in register 1. You must then move the record to a work area. You can process the record either before or after the move.

The PUT macro instruction (step B, Figure 34) specifies the address of the work area containing the next output record. The system returns the address of the next output buffer available for use as a work area in register 1. That address is passed to the move (MVC) instruction in register 6.

The GET macro instruction (step C, Figure 34) locates the next input record. You must then move the record to the new work area. Notice that the previous work area becomes part of the output buffer (step C).

Note that if records other than format–F records are being moved, the length attribute of the MVC instruction must be changed as shown by the code beginning with the USING statement in Figure 34. If the record is more than 256 bytes long, you must code a move routine to process the complete record.




```
            LA      6,WORKAREA
NEXTREC     GET     INDCB
            LR      7,1
            USING   IHADCB,5
            LA      5,INDCB
            LH      4,DCBLRECL
            SH      4,=H'1'
            EX      4,MOVEREC
            ...
            PUT     OUTDCB,(6)
            LR      6,1
            B       NEXTREC
MOVEREC     MVC     0(1,6),0(7)
WORKAREA    DS      CL50
INDCB       DCB     MACRF=(GL),---
OUTDCB      DCB     MACRF=(PT),---
            DCBD    DSORG=(LR)
```

Figure 34. Exchange Buffering with MACRF=GL and MACRF=PT

Buffering Techniques and GET/PUT Processing Modes: As you can see from the previous examples, the most efficient code is achieved by use of automatic buffer pool construction, and GET–locate and PUTX–output with either simple or exchange buffering. Figure 35 summarizes the combinations of buffering techniques and processing modes that can be used. Notice, for example, that if you use PUT–locate and GET–substitute, you must provide a work area and you must also move the record from the work area to the output buffer.

RELSE — Release an Input Buffer

When using the queued access technique to process a sequential or indexed sequential data set, you can direct the system to ignore the remaining records in the input buffer being processed. The next GET macro instruction retrieves a record from another buffer. If format–V spanned records are being used, the next logical record obtained may begin on any segment in any subsequent block.

If you are using move mode, the buffer is made available for refilling as soon as the RELSE macro instruction is issued. When you are using locate mode, the system does not refill the buffer until the next GET macro instruction is issued. If a PUTX macro instruction has been used, the block is written before the buffer is refilled.

Output Buffering: →	Simple		Exchange		Simple		Exchange		Simple	Input Buffering: Exchange	Simple		Exchange		Simple		Exchange	
Input Buffering: → Simple Actions ↓	GET-move, PUT-locate	GET-move, Put-move	GET-move, PUT-move	GET-move PUT-substitute	GET-locate, PUT-locate	GET-locate, PUT-move	GET-locate, PUT-move	GET-locate, PUT-substitute	GET-locate (logical record), PUT-locate		GET-locate, PUT-locate	GET-locate, PUT-move	GET-locate, PUT-move	GET-locate, PUT-substitute	GET-substitute, PUT-locate	GET-substitute, PUT-move	GET-substitute, PUT-move	GET-substitute, PUT-substitute
Program must move record					X			X	X		X			X	X			
System moves record	X	X	X	X		X	X					X	X			X	X	
System moves record segment									X									
Record is not moved													X[1]					X
Work area required		X	X	X				X						X	X	X	X	X
PUTX - output can be used						X	X					X	X					

[1] PUTX, only

Figure 35. Buffering Technique and GET/PUT Processing Modes

TRUNC — Truncate an Output Buffer

When using the queued access technique to process a sequential data set, you can direct the system to write a short block. The first record in the next buffer is the next record processed by a PUT–output or PUTX–output mode.

If the locate mode is being used, the system assumes that a record has been placed in the buffer segment pointed to by the last PUT macro instruction.

The last block of a data set is truncated by the Close routine. Note that a data set containing format–F records with truncated blocks cannot be read from direct–access storage as efficiently as a standard format–F data set.

GETBUF — Get a Buffer from a Pool

The GETBUF macro instruction can be used with the basic access technique to request a buffer from a buffer pool constructed by the BUILD, GETPOOL, or OPEN macro

instruction. The address of the buffer is returned by the system in a register you specify when you issue the macro instruction. If no buffer is available, the register contains 0 instead of an address.

FREEBUF — Return a Buffer to a Pool

The FREEBUF macro instruction is used with the basic access technique to return a buffer to the buffer pool from which it was obtained by a GETBUF macro instruction. Although the buffers need not be returned in the order in which they were obtained, they must be returned when they are no longer needed.

FREEDBUF — Return a Dynamic Buffer to a Pool

Any buffer obtained through the dynamic buffer option must be released before it can be used again. When you are processing a direct data set, if you do not update the block in the buffer and thus free the buffer when the block is written, you must use the FREEDBUF macro instruction. If there is an uncorrectable input/output error, the control program releases the buffer. When you are processing an indexed sequential data set, if you do not update the block in the buffer or if there is an uncorrectable input error, the control program releases the buffer when the next READ macro instruction is issued on the same DECB, unless you use the FREEDBUF macro instruction.

To effect the release, you must specify the address of the DECB that was created when the block was read using the dynamic buffering option, as well as the address of the DCB associated with the data set being processed.

Processing a Sequential Data Set

Data sets residing on all volumes other than direct–access volumes must be processed sequentially. In addition, a data set residing on a direct–access volume, regardless of organization, can be processed sequentially. This feature of the operating system allows you to write your program with little regard for the type of device to be used when the program is executed, except for restrictions on the use of certain device–dependent macro instructions and processing options.

Either the queued or the basic access technique may be used to store and retrieve the records of a sequential data set. Additionally, a technique called *chained scheduling* can be used to accelerate the input/output operations required for a sequential data set.

Data Format — Device Type Considerations

Before execution of your program, you must supply OS with both the record format (RECFM) and device–dependent (DEVD) information in a DCB macro instruction, a DD statement, or a data set label. The DCB subparameters for the DD statement differ slightly from those described here. A complete description of the DD statement and a glossary of DCB subparameters is contained in *OS Job Control Language Reference.*

The record format (RECFM) parameter of the DCB macro instruction specifies the characteristics of the records in the data set as fixed–length (RECFM=F), variable–length (RECFM=V or D), or undefined–length (RECFM=U). Fixed–length blocked records (RECFM=FB) can be specified as standard (RECFM=FBS), which means there are no truncated (short) blocks or unfilled tracks within the data set, with

the possible exception of the last block or track. Data sets with a fixed–length, standard format were described previously under "Fixed–Length Records, Standard Format."

As an optional feature, a control character can be contained in each record. This control character will be recognized and processed if the data set is printed or punched. The control characters are transmitted on both tapes and direct–access volumes. The presence of a control character is indicated by M or A in the RECFM field of the data control block. M denotes machine code; A denotes American National Standards Institute (ANSI) code. If either M or A is specified, the character must be present in every record; the printer space (PRTSP) or stacker select (STACK) field of the DCB is ignored. The optional control character must be in the first byte of format–F and format–U records and in the fifth byte of format–V records and format–D records where BUFOFF=L. Control character codes are listed in "Appendix B: Control Characters."

The device–dependent (DEVD) parameter of the DCB macro instruction specifies the type of device on which the data set's volume resides:

TA magnetic tape
PT paper tape reader
PR printer
PC card punch
RD card reader
DA direct–access device

Magnetic Tape (TA)

Format–F, –V, –D, and –U records are acceptable for magnetic tape. Format–V records are not acceptable on 7–track tape if the data conversion feature[1] is not available. ASCII records are not acceptable on 7–track tape.

When you create a tape data set with variable–length record format (V or D), the control program pads any data block shorter than 18 bytes. For format–V records, it pads to the right with binary 0s so that the data block length equals 18 bytes. For format–D (ASCII) records, the padding consists of ASCII circumflex characters.

Note that there is no minimum requirement for blocksize; however, if a data check occurs on a magnetic–tape device, any record shorter than 12 bytes in a read operation or 18 bytes in a write operation will be treated as a noise record and lost. No check for noise will be made unless a data check occurs.

Tape density (DEN) specifies the recording density in bits per inch per track, as shown in Figure 36. If this information is not supplied, the highest applicable density is assumed.

The track recording technique (TRTCH) for 7–track tape can be specified as:

C Data conversion is to be used.

E Even parity is to be used; if E is omitted, odd parity is assumed.

T BCDIC to EBCDIC translation is required.

[1] Data conversion makes it possible to write 8 binary bits of data on 7 tracks. Otherwise, only 6 bits of an 8–bit byte are recorded. The length field of format–V records contains binary data and is not recorded correctly without data conversion.

DEN Value	Recording Density Models 2400 and 3400			
	7–Track[1]	9–Track	9–Track (phase encoded)	9–Track (dual–density)
0	200	—	—	—
1	556	—	—	—
2	800	800	—	800[2]
3	—	—	1600	1600[3]

[1] No 7–track feature on the 3410
[2] Non–return–to–zero–inverse (NRZI) mode
[3] Phase encoding (PE) mode

Figure 36. Tape Density (DEN) Values

Paper–Tape Reader (PT)

The paper–tape reader accepts format–F and format–U records. If you use QSAM, you should not specify the records as blocked. Each format–U record is followed by an end–of–record character. Data read from paper tape may optionally be converted into the System/360 or System/370 internal representation of one of six standard paper–tape codes. Any character found to have a parity error will not be converted when the record is transferred into the input area. Characters deleted in the conversion process are not counted in determining the blocksize.

The following symbols indicate the code in which the data was punched. If this information is omitted, I is assumed.

I IBM BCD perforated tape and transmission code (8 tracks)
F Friden (8 tracks)
B Burroughs (7 tracks)
C National Cash Register (8 tracks)
A ASCII (8 tracks)
T Teletype (5 tracks)
N No conversion

Note that when you are using QSAM, the processing mode must be move mode.

Card Reader and Punch (RD/PC)

Format–F, –V, and –U records are acceptable to both the reader and punch. The device control character, if specified in the RECFM parameter, is used to select the stacker; it is not punched. The first 4 bytes (record descriptor word or segment descriptor word) of format–V records or record segments are not punched. For format–V records, at least 1 byte of data must follow the record or segment descriptor word or the carriage control character.

Each punched card corresponds to one physical record. Therefore, you should restrict the maximum record size to 80 (EBCDIC mode) or 160 (column binary mode) data bytes. When mode (C) is used for the card punch, BLKSIZE must be 160 unless you are using PUT. Then you can specify BLKSIZE as 160 or a multiple of 160, and the system handles this as described earlier under "PUT — Write a Record" in the section "Queued Access Techniques." You can specify the read/punch mode of operation (MODE) parameter as either card image (column binary) mode (C) or EBCDIC mode (E). If this information is omitted, E is assumed.

The stacker selection parameter (STACK) can be specified as either 1 or 2 to indicate which bin is to receive the card. If it is not specified, 1 is assumed.

Note that when QSAM is used, punch error correction on the IBM 2540 Card Read Punch is automatically performed only for data sets using three or more buffers without the chained scheduling feature.

The 3525 Card Punch accepts only format–F records for print data sets and for associated data sets. Other record formats are allowed for the read data set, the punch data set, and the interpret punch data set. See "Appendix C: Special Programming Considerations for the 3505 Card Reader and the 3525 Card Punch" for more information on programming for the 3525 Card Punch.

Printer (PR)

Records of format F, V, and U are acceptable to the printer. The first 4 bytes (record descriptor word or segment descriptor word) of format–V records or record segments are not printed. For format–V records, at least 1 byte of data must follow the record or segment descriptor word or the carriage control character. The carriage control character, if specified in the RECFM parameter, is not printed. However, the system does not position the printer to channel 1 for the first record.

Because each line of print corresponds to one record, the record length should not exceed the length of one line on the printer. For variable–length spanned records, each line corresponds to one record segment, and blocksize should not exceed the length of one line on the printer.

If carriage control characters are not specified, you can indicate printer spacing (PRTSP) as 0, 1, 2, or 3. If it is not specified, 1 is assumed.

Direct–Access Device (DA)

Direct–access devices accept records of format F, V, or U. If the records are to be read or written with keys, the key length (KEYLEN) must be specified. In addition, the operating system has a standard track format for all direct access volumes. Each track contains data information as well as certain control information such as:

- The address of the track
- The address of each record
- The length of each record
- Gaps between areas

A complete description of track format is contained in the section "Direct–Access Device Characteristics." Your only concern in creating a sequential data set is to allow for an 8–byte track descriptor record (capacity record or R0) when requesting space on a direct–access volume. In addition, device overhead, which varies with the device, must be allocated for each block on the track.

Device Control

The operating system provides you with six macro instructions for controlling input/output devices. Each is, to varying degrees, device–dependent. Therefore, you must exercise some care if you wish to achieve device independence.

When you use the queued access technique, only unit record equipment can be controlled directly. When using the basic access technique, limited device independence can be achieved between magnetic–tape and direct–access devices. You must check all read or write operations before issuing a device control macro instruction.

CNTRL — Control an I/O Device

The CNTRL macro instruction performs these device–dependent control functions:

- Card reader stacker selection (SS)
- Printer line spacing (SP)
- Printer carriage control (SK)
- Magnetic–tape backspace (BSR) over a specified number of blocks
- Magnetic–tape backspace (BSM) past a tapemark and forward space over the tapemark
- Magnetic–tape forward space (FSR) over a specified number of blocks
- Magnetic–tape forward space (FSM) past a tapemark and a backspace over the tapemark

Backspacing moves the tape toward the load point; forward spacing moves the tape away from the load point.

Note that the CNTRL macro instruction cannot be used with an input data set containing variable–length records on the card reader.

You can use the CNTRL macro instruction to position DOS tapes that contain embedded DOS checkpoint records if you specify OPTCD=H in the DCB parameter field of the DD statement. The CNTRL macro instruction cannot be used to backspace DOS 7–track tapes that are written in data convert mode and contain embedded checkpoint records.

PRTOV — Test for Printer Overflow

The PRTOV macro instruction tests for channel 9 or 12 of the printer carriage control tape or the forms control buffer (FCB). An overflow condition causes either an automatic skip to channel 1 or, if specified, transfer of control to your routine for overflow processing. If you specify an overflow exit routine, set DCBIFLGS to X'00' before issuing another PRTOV.

If the data set specified in the DCB is not for a printer, no action is taken.

SETPRT — Load Universal Character Set and Forms Control Buffers

The SETPRT macro instruction indicates the character set to be used by a printer with the Universal Character Set (UCS) feature. When issued, SETPRT loads the UCS buffer with a character set image from the image library. The image library is a cataloged data set containing the UCS and FCB images for the 1403 and 3211 printers. It is located in the system library called SYS1.IMAGELIB. Thus, it allows your program to change character sets during execution. SETPRT also allows you to request the operator to verify loading of the buffer and to specify the printing of

lowercase EBCDIC characters in uppercase when no uppercase/lowercase print chain or train is available.

For a printer that has no carriage control tape, you can use the SETPRT macro instruction to load the FCB, to request operator verification of buffer loading, and to allow the operator to align the paper in the printer.

The FCB contents can be fetched from the system library or defined in your program through the exit list of the DCB macro instruction, as discussed under "Exit List (EXLST)."

When issued, the SETPRT macro instruction loads a special UCS buffer from the system library. The library contains images of standard IBM character sets and of your own special character sets. The operator can be requested to verify the loaded image after mounting the appropriate print chain or train.

The SETPRT macro instruction can be used to block or deblock printer data checks. When data checks are blocked, unprintable characters are treated as blanks and do not cause an error condition.

BSP — Backspace a Magnetic–Tape or Direct–Access Volume

The BSP macro instruction backspaces one block on the magnetic–tape or direct–access volume being processed. The block can then be reread or rewritten. An attempt to rewrite the block destroys the contents of the remainder of the tape or track.

The direction of movement is toward the load point or beginning of the extent. You may not use the BSP macro instruction if the track overflow option was specified or if the CNTRL, NOTE, or POINT macro instruction is used. The BSP macro instruction should be used only when other device control macro instructions could not be used for backspacing.

You can use the BSP macro instruction to backspace DOS tapes containing embedded DOS checkpoint records. If you use this means of backspacing, you must test for and bypass the embedded checkpoint records. You cannot use the BSP macro instruction for DOS 7–track tapes written in translate mode.

NOTE — Return the Relative Address of a Block

The NOTE macro instruction requests the relative address of the block just read or written. In a multivolume data set, the address is relative to the beginning of the volume currently being processed.

The address provided by the operating system is returned in register 1. The address is in the form of a 4–byte relative block address for magnetic tape; for a direct–access device, it is a 4–byte relative track address. The amount of unused space available on the track of the direct–access device is returned in register 0.

POINT — Position to a Block

The POINT macro instruction causes repositioning of a magnetic–tape or direct–access volume to a specified block. The next read or write operation begins at this block. In a multivolume data set, you must ensure that the volume referred to is the volume currently being processed. If a write operation follows the POINT macro instruction, all of the track following the write operation is erased unless the data set is opened for UPDAT.

You can use the POINT macro instruction to position DOS tapes that contain embedded checkpoint records if you specify OPTCD=H in the DCB parameter field of the DD statement. The POINT macro instruction cannot be used to backspace DOS 7–track tapes that are written in data convert mode and contain embedded checkpoint records.

Device Independence

The ability to request input/output operations without regard for the physical characteristics of the I/O devices makes it possible for you to write one program that will fulfill a variety of needs. Device independence may be useful for:

- Accepting data from a number of recording devices, such as 1316 disk pack, 7– or 9–track magnetic tape, or unit–record equipment. This situation could arise when several types of data–acquisition devices are feeding a centralized complex.
- Observing constraints imposed by the availability of input/output devices (for example, when devices on order have not been installed).
- Assembling, testing, and debugging on one System/360 or System/370 configuration and processing on a different configuration. For example, a 2311 drive can be used as a substitute for several magnetic–tape units.

Device independence is not difficult to achieve, but requires some planning and forethought. There are two areas of planning necessary to achieve device independence — system generation considerations and programming considerations.

System Generation Considerations

You can provide for device independence when the system is generated by generating a system that not only meets the current input/output configuration requirements but includes anticipated device attachments. Creating such a system entails looking ahead at expected delivery of input/output devices and, during system generation, constructing the necessary control blocks and tables. Thus, when the devices are delivered, they need only be physically attached. The operating system recognizes the devices without modification. During the interim, unconnected devices must be placed off–line by a VARY command issued by the operator.

When new device attachments cannot be fully anticipated, you can add new devices by performing an I/O device generation. This is a limited type of system generation that enables you to change your I/O configuration without regenerating other parts of the system.

System generation techniques for effecting a smooth transition to new input/output devices do not include addition of new device types. When support for new devices is provided, a new system must be generated. A complete description of system generation techniques is contained in *OS System Generation.*

Programming Considerations

Each of three data set organizations — partitioned, indexed sequential, and direct — requires the use of a direct–access device. Device independence is meaningful, then, only for a sequentially organized data set, that is, a data set where one block of data follows another, thus allowing input or output to be on a magnetic tape drive, a direct–access device, a card read/punch, or a printer.

Your program will be device–independent if you do two things:

- Omit all device–dependent macro instructions and macro instruction parameters from your program.
- Defer specifying any required device–dependent parameters until the program is ready for execution. That is, supply the parameters on your data definition (DD) statement.

In examining the following list of macro instructions, consider only the logical layout of your data record without regard for the type of device used. Also, consider that any reference to a direct–access volume is to be treated like a reference to magnetic tape, that is, you must create a new data set rather than attempt to update.

OPEN

Specify INPUT, OUTPUT, INOUT, or OUTIN. The parameters RDBACK and UPDATE are device–dependent and cause an abnormal termination if directed to a device of the wrong type.

READ

Specify forward reading (SF) only.

WRITE

Specify forward writing (SF) only; use only to create new records.

PUTX

Use only output mode.

NOTE/POINT

These macros are valid for both magnetic–tape and direct–access volumes.

BSP

This macro is valid for magnetic–tape or direct–access volumes. However, its use would be an attempt to perform device–dependent action.

CNTRL/PRTOV

These macros are device–dependent.

DCB Subparameters

MACRF

Specify R/W or G/P. Processing mode can also be indicated.

DEVD

Specify DA if any direct–access device may be used. Magnetic–tape and unit–record equipment DCBs will fit in the area provided during assembly. Specify unit–record devices only if you expect never to change to tape or direct–access devices. Key length (KEYLEN) can be specified on the DD statement if necessary.

RECFM, LRECL, BLKSIZE

These can be specified in the DD statement. However, you must consider maximum record size for specific devices, and track overflow cannot be specified unless supported.

DSORG

Specify sequential organization (PS or PSU).

OPTCD

This subparameter is device–dependent; specify it in the DD statement.

SYNAD

Any device–dependent error checking is automatic. Generalize your routine so that no device–dependent information is required.

Chained Scheduling for I/O Operations

To accelerate the input/output operations required for a data set, the operating system provides a technique called *chained scheduling*. When requested, the system bypasses the normal I/O routines and dynamically chains several input/output operations together. A series of separate read or write operations, functioning with chained scheduling, is issued to the computing system as one continuous operation. The program–controlled interruption (PCI) flag in the CCWs is used for synchronization of the I/O operations.

The I/O performance is improved by reduction in both the CPU time and the channel start/stop time required to transfer data between main and auxiliary storage. The effects of rotational delay are also reduced since several successive blocks, requested separately, can be retrieved in a single rotation. Chained scheduling can be used only with simple buffering. Each data set for which chained scheduling is specified must be assigned at least two, and preferably three, buffers.

A request for chained scheduling will be ignored and normal scheduling used if any of the following are encountered when the data set is opened:

- BDAM CREATE, that is, MACRF=(WL)
- Track overflow
- UPDAT in the operand of the OPEN macro instruction
- Exchange buffering
- CNTRL macro instruction to be used
- Device type is paper tape reader
- A print data set or any associated data set for the 3525 Card Punch. (See "Appendix C: Special Programming Considerations for the 3505 Card Reader and the 3525 Card Punch" for more information on programming for the 3525.)
- Bypassing of embedded DOS checkpoint records on tape input data sets

When chained scheduling is being used, the automatic skip feature of the PRTOV macro instruction for the printer will not function. Format control must be achieved by

ANSI or machine control characters. (Control characters are discussed in more detail in Part 1 under "Control Character," in Part 2 under "Data Format — Device Type Considerations," and in "Appendix B: Control Character.") When you use undefined–length records with QSAM, the DCBLRECL field is equal to the BLKSIZE field, not the actual record length. The entire block is moved to your work area in the move mode. In locate mode, the address of the beginning of the block is returned in register 1.

When chained scheduling is used on the 2540 Card Read Punch, error recovery procedures are not performed.

Chained scheduling is most valuable for programs that require extensive input and output operations. Because a data set using chained scheduling may monopolize available time on a channel, separate channels should be assigned, if possible, when more than one data set is to be processed.

Search Direct for Input Operations

To accelerate the input operations required for a data set, the operating system provides a technique called *search direct.* Search direct reads in the requested record and the count field of the second record. This allows the operation to get the next record directly, along with the count field of the following record. Search direct can be used with all record formats except format–UT, format–FBT, and spanned. You request search direct by coding OPTCD=Z in the DCB macro instruction except for record formats FS and FBS. It is an automatic feature for formats FS and FBS. This technique cannot be used with the NOTE and POINT macro instructions when you specify the UPDAT option of OPEN.

```
            ...
            OPEN      (INDATA,,OUTDATA,(OUTPUT))
NEXTREC     GET       INDATA,WORKAREA              Move mode
            AP        NUMBER,=P'1'
            UNPK      COUNT,NUMBER                 Record count adds 6
            PUT       OUTDATA,COUNT                bytes to each record
            B         NEXTREC
TAPERROR    SYNADAF   ACSMETH=QSAM                 Control program returns message
            LA        0,68(0,1)                    address in register 1.
            ST        14,SAVE14                    SYNAD routine prints part of
            PUT       OUTDATA,(0)                  the message (beginning with
            SYNADRLS                               the unit number) as a 56-byte
            L         14,SAVE14                    fixed-length record.  It then
            RETURN                                 returns to the control
ENDJOB      CLOSE     (INDATA,,OUTDATA)            program.
            ...
COUNT       DS        CL6
WORKAREA    DS        CL50
NUMBER      DC        PL4'0'
SAVE14      DS        F
INDATA      DCB       DDNAME=INPUTDD,DSORG=PS,MACRF=(GM),EROPT=ACC,                C
                  SYNAD=TAPERROR,EODAD=ENDJOB
OUTDATA     DCB       DDNAME=OUTPUTDD,DSORG=PS,MACRF=(PM),EROPT=ACC
```

Figure 37. Creating a Sequential Data Set — Move Mode, Simple Buffering

Creating a Sequential Data Set

As discussed earlier, a processing program should be developed using, as much as possible, factors that are constant, with variable factors specified at execution. For that reason, the following examples are generalized as much as possible. They are neither exhaustive nor intended as complete examples. Rather, they are presented as introductory sequences.

Since the basic access technique for sequential processing is usually used to create a partitioned data set or a direct data set, examples of the READ and WRITE macro instructions are deferred for discussion in those areas. There is no other reason, however, for them not to be used in place of the queued access macro instructions if automatic blocking and anticipatory buffering are not required.

Tape–to–Print, Move Mode — Simple Buffering: In Figure 37, the GET–move and PUT–move require two movements of the data records. If the record length (LRECL) does not change in processing, only one move is necessary; you can process the record in the input buffer segment. A GET–locate can be used to provide a pointer to the current segment.

Tape–to–Print, Locate Mode — Simple Buffering: This example (Figure 38) is similar to that in Figure 37. However, since there is no change in the record length, the records can be processed in the input buffer. Only one move of each data record is required.

```
          ...
          OPEN     (INDATA,,OUTDATA,(OUTPUT),ERRORDCB,(OUTPUT))
NEXTREC   GET      INDATA                          Locate mode
          LR       2,1                             Save pointer
          AP       NUMBER,=P'1'
          UNPK     0(6,2),NUMBER                   Process in input area
          PUT      OUTDATA                         Locate mode
          MVC      0(50,1),0(2)                    Move record to output buffer
          B        NEXTREC
TAPERROR  SYNADAF  ACSMETH=QSAM                    Message address in register 1
          ST       2,SAVE2                         Save register 2 contents
          L        2,8(0,1)                        Load pointer to input buffer
          MVC      8(70,1),50(1)                   Shift nonblank message fields
          MVC      78(50,1),0(2)                   Add input record to message
          LA       0,4(1)                          Load address of message
          LR       2,14                            Save return address
          PUT      ERRORDCB,(0)                    Print message (move mode)
          SYNADRLS                                 Release message and save area
          LR       14,2                            Restore return address
          L        2,SAVE2                         Restore register 2 contents
          RETURN                                   Return to control program
ENDJOB    CLOSE    (INDATA,,OUTDATA,,ERRORDCB)
          ...
NUMBER    DC       PL4'0'
INDATA    DCB      DDNAME=INPUTDD,DSORG=PS,MACRF=(GL),EROPT=ACC,                C
               SYNAD=TAPERROR,EODAD=ENDJOB
OUTDATA   DCB      DDNAME=OUTPUTDD,DSORG=PS,MACRF=(PL),EROPT=ACC
ERRORDCB  DCB      DDNAME=SYSOUTDD,DSORG=PS,MACRF=(PM),RECFM=V,                 C
               BLKSIZE=128,LRECL=124,EROPT=ACC
SAVE2     DS       F
```

Figure 38. Creating a Sequential Data Set — Locate Mode, Simple Buffering

Tape–to–Print, Substitute Mode — Exchange Buffering: Although the initial problem is the same, the solution described in Figure 39 takes advantage of exchange buffering, which eliminates the need to move the data record, and makes direct reference to individual fields within a record through the use of a dummy control section (DSECT). The use of the DSECT allows symbolic reference to be made for storage–to–storage operations; therefore, the length attributes need not be explicitly stated.

```
            ...
            OPEN      (INDATA,,OUTDATA,(OUTPUT),ERRORDCB,(OUTPUT))
            LA        0,GIVEAWAY                    Set up for first buffer
NEXTREC     GET       INDATA,(0)                    Substitute mode
            LR        2,1                           Pointer to next record
            USING     RECORD,2                      Establish address of DSECT
            AP        NUMBER,=P'1'
            UNPK      COUNT,NUMBER
            PUT       OUTDATA,RECORD                Substitute mode
            LR        0,1                           Exchange work area
            B         NEXTREC
TAPERROR    SYNADAF   ACSMETH=QSAM                  SYNAD routine is same
            ...                                     as in previous example
ENDJOB      CLOSE     (INDATA,,OUTDATA,,ERRORDCE)
            ...
            DS        0D
GIVEAWAY    DS        CL50
NUMBER      DC        PL4'-'
INDATA      DCB       DDNAME=INPUTDD,DSORG=PS,MACRF=(GT),BFTEK=E,BFALN=D,        C
                  EROPT=ACC,SYNAD=TAPERROR,EODAD=ENDJOB
OUTDATA     DCB       DDNAME=OUTPUTDD,DSORG=PS,MACRF=(PT),BFTEK=E,BFALN=D,       C
                  EROPT=ACC
            ...
RECORD      DSECT
COUNT       DS     CL6
RESTOFIT    DS     CL44
```

Figure 39. Creating a Sequential Data Set — Substitute Mode, Exchange Buffering

Using BSAM to Read Fixed–Length Blocked Records

When you read a sequential data set, you can determine the length of the record in one of the following three ways, depending upon the record format of the data set:

- For fixed–length records, the length of all records is the value in the DCBBLKSI field of the DCB.
- For variable–length records, the block descriptor word in the record contains the length of the record.
- For fixed–length blocked or undefined–length records, the following method can be used to calculate the block length. After checking the DECB for the READ request but before issuing any subsequent data management macro instructions that specify the DCB for the READ request, obtain the IOB address from the DECB. The IOB address can be loaded from the location 16 bytes from the start of the DECB.

Obtain the residual count from the channel status word (CSW) that has been stored in the input/output block (IOB). The residual count is in the halfword 14 bytes from the start of the IOB. Subtract this residual count from the number of data bytes requested to be read by the READ macro instruction. If "S" was coded as the length parameter

of the READ macro instruction or the length parameter was omitted, the number of bytes requested is the value of DCBBLKSI at the time the READ was issued. If the length was coded in the READ macro instruction, this value is the number of data bytes and it is contained in the halfword 6 bytes from the beginning of the DECB. The result of the subtraction is the length of the block read. See Figure 40.

```
         OPEN     (DCB,(INPUT))
         LA       DCBR,DCB
         USING    IHADCB,DCBR
         ...
         READ     DECB1,SF,DCB,AREA1
         READ     DECB2,SF,DCB,AREA2,50
         ...
         CHECK    DECB1
         LH       WORK1,DCBBLKSI          Blocksize at time of READ
         L        WORK2,DECB1+16          IOB address
         SH       WORK1,14(WORK2)         WORK1 has block length
         ...
         CHECK    DECB2
         LH       WORK1,DECB2+6           Length requested
         L        WORK2,DECB2+16          IOB address
         SH       WORK1,14(WORK2)         WORK1 has block length
         ...
         MVC      DCBBLKSI,LENGTH3        Length to be read
         READ     DECB3,SF,DCB,AREA3
         ...
         CHECK    DECB3
         LH       WORK1,LENGTH3           Blocksize at time of READ
         L        WORK2,DECB+16           IOB address
         SH       WORK1,14(WORK2)         WORK1 has block length
         ...
DCB      DCB      ...RECFM=U,NCP=2,...
         DCBD
```

Figure 40. Using BSAM to Read Fixed–Length Blocked Records

Processing a Partitioned Data Set

A partitioned data set can be stored only on a direct–access device. It is divided into sequentially organized *members,* each made up of one or more records (see Figure 41). Each member has a unique name, 1 to 8 characters long, stored in a *directory* that is part of the data set. The records of a given member are stored or retrieved sequentially.

The main advantage of using a partitioned data set is that you can retrieve any individual member once the data set is opened without searching the entire data set. For example, a program library can be stored as a partitioned data set, each member of which is a separate program or subroutine. The individual members can be added or deleted as required. When a member is deleted, the member name is removed from the directory, but the space used by the member cannot be reused until the data set is reorganized.

The directory, a series of records at the beginning of the data set, contains an entry for each member. Each directory entry contains the member name and the starting location of the member within the data set, as shown in Figure 41. In addition, you

can specify up to 62 characters of information in the entry. The directory entries are arranged in alphameric collating sequence by name.

The track address of each member is recorded by the system as a relative track address within the data set rather than as an absolute track address. Thus, an entire data set can be moved without changing the relative track addresses in the directory. The data set can be considered as one continuous set of tracks regardless of how the space was actually allocated.

If there is not sufficient space available in the directory for an additional entry, or not enough space available within the data set for an additional member, no new members can be stored.

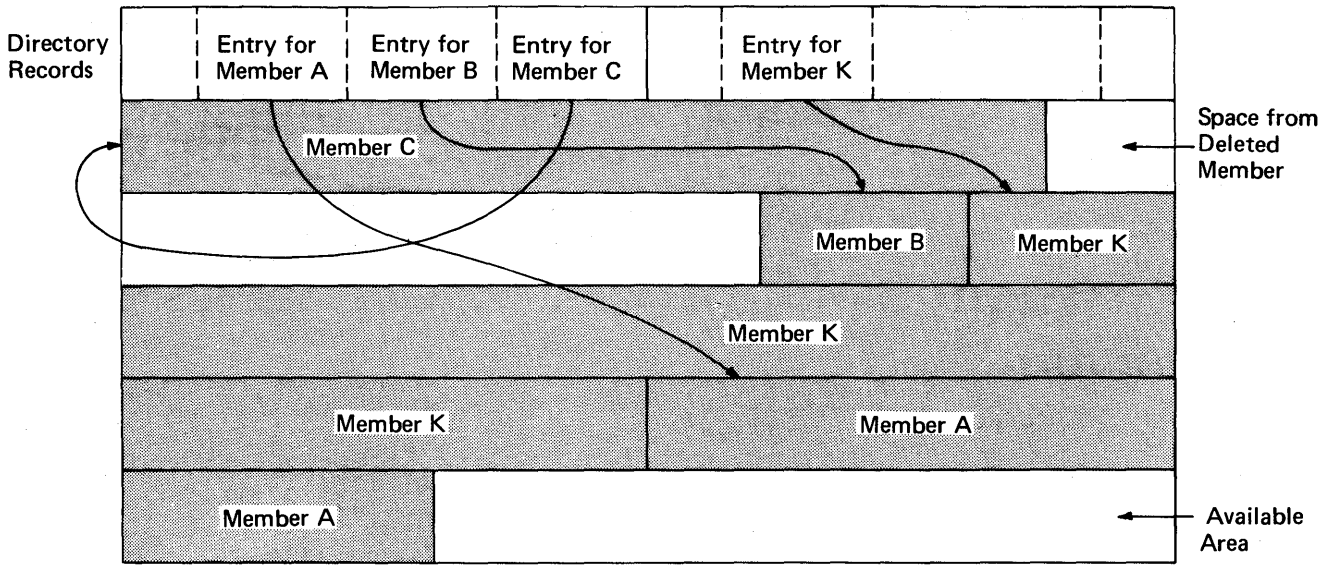


Figure 41. A Partitioned Data Set

Partitioned Data Set Directory

The directory of a partitioned data set occupies the beginning of the area allocated to the data set on a direct–access volume. It is searched and maintained by the FIND and STOW macro instructions. The directory consists of member entries arranged in ascending order according to the binary value of the member name or alias.

Member entries vary in length and are blocked into 256–byte blocks. Each block contains as many complete entries as will fit in a maximum of 254 bytes; any remaining bytes are left unused and are ignored. Each directory block contains a 2–byte count field that specifies the number of active bytes in a block. As shown in Figure 42, each block is preceded by a hardware–defined key field containing the name of the last member entry in the block, that is, the member name with the highest binary value.

Each member entry contains a member name or alias. There can be as many as 16 aliases for each member. Each entry also contains the relative track address of the member and a count field, as shown in Figure 43. In addition, it may contain a user data field. The last entry in the last directory block has a name field of maximum binary value — all 1s.

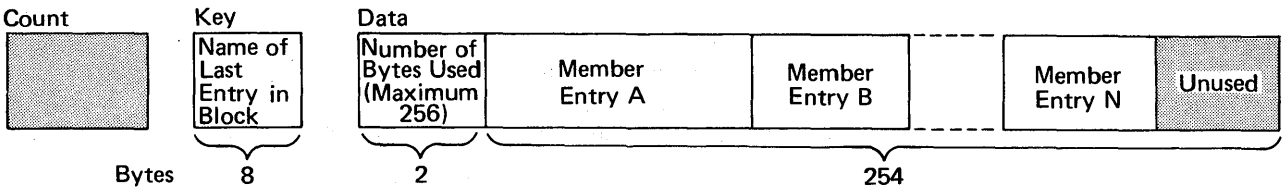


Figure 42. A Partitioned Data Set Directory Block

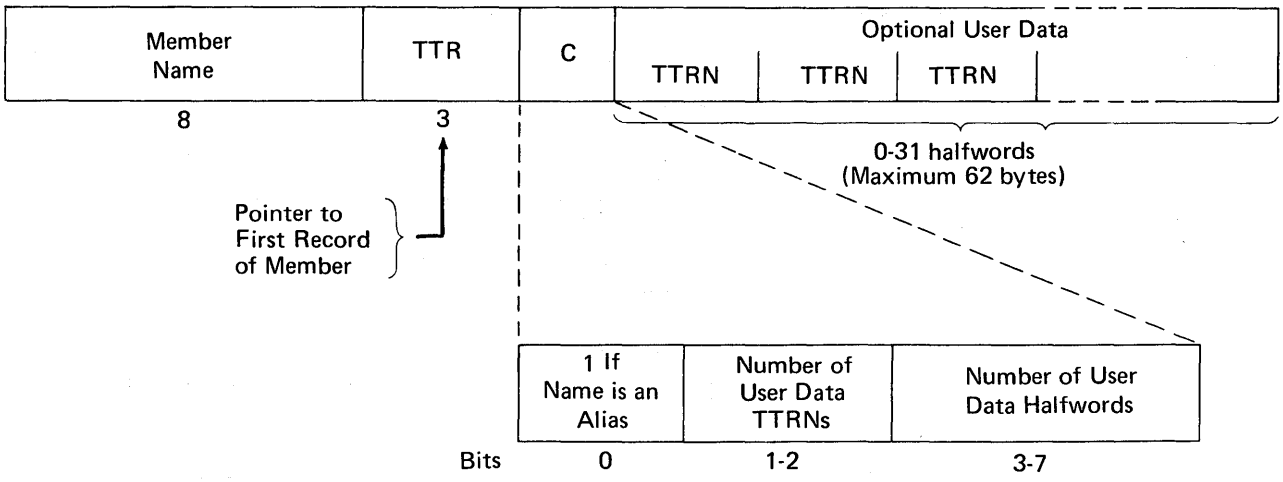


Figure 43. A Partitioned Data Set Directory Entry

NAME

specifies the member name or alias. It contains up to 8 alphameric characters, left–justified and padded with blanks if necessary.

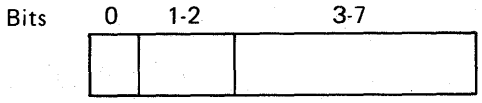
TTR

is a pointer to the first block of the member; TT is the number of the track, relative to the beginning of the data set, and R is the number of the block, relative to the beginning of that track.

Note: This pointer is created by adding 1 to the TTR for the last block of the previous member (which is an end–of–file mark). If track TT is full, the next block will begin at record 1 of track TT + 1, and the pointer will be updated accordingly. The control program finds the block by searching in multitrack mode using TT(R–1) as a search argument.

C

specifies the number of halfwords contained in the user data field. It may also contain additional information about the user data field, as shown below:



0 when set to 1, indicates that the NAME field contains an alias.

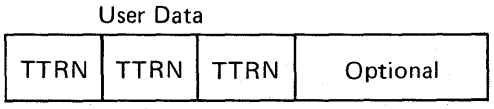
1–2 specifies the number of pointers to locations within the member.

A maximum of three pointers is allowed in the user data field. Additional pointers may be contained in a record referred to as a note list, discussed below. The pointers can be updated automatically if the data set is moved or copied by a utility program such as IEHMOVE. The data set must be marked unmovable under the following conditions:

- More than three pointers are used in the user data field.
- The pointers in the user data field or note list do not conform to the standard format.
- The pointers are not placed first in the user data field.
- Any direct access address (absolute or relative) are embedded in any data blocks or in another data set that refers to this data set.

3–7 contains a binary value indicating the number of halfwords of user data. This number must include the space used by pointers in the user data field.

You can use the user data field to provide variable data as input to the STOW macro instruction. If pointers to locations within the member are provided, they must be 4 bytes long and placed first in the user data field. The user data field format is as follows:



TT is the relative track address of the note list or area to which you are pointing.

R is the relative block number on that track.

N is a binary value that indicates the number of additional pointers contained in a note list pointed to by the TTR. If the pointer is not to a note list, N=0.

A note list consists of additional pointers to blocks within the same member of a partitioned data set. You can divide a member into subgroups and store a pointer to the beginning of each subgroup in the note list. The member may be a load module containing many control sections (CSECTs), each CSECT being a subgroup pointed to by an entry in the note list. You get the pointer to the beginning of the subgroup by using the NOTE macro instruction after you write the first record of the subgroup. Remember that the pointer to the first record of the member is stored in the directory entry by the system.

If the existence of a note list was indicated as shown above, the list can be updated automatically when the data set is moved or copied by a utility program such as IEHMOVE. Each 4–byte entry in the note list has the following format:

TT is the relative track address of the area to which you are pointing.

R is the relative block number on that track.

X is available for any use.

To place the note list in the partitioned data set, you must use the WRITE macro instruction. After checking the write operation, use the NOTE macro instruction to determine the address of the list and place that address in the user data field of the directory entry.

Processing a Member of a Partitioned Data Set

Because a member of a partitioned data set is sequentially organized, it is processed in the same manner as a sequential data set. Either the basic or queued access technique can be used. However, you cannot alter the directory when using the queued technique.

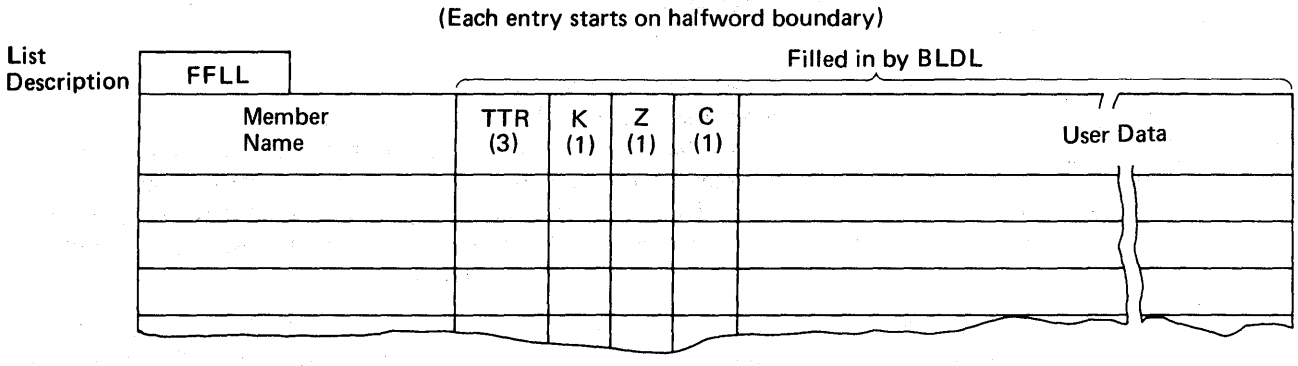
To locate a member or to process the directory, several macro instructions are provided by the operating system. The BLDL macro instruction can be used to structure a list of directory entries in main storage; the FIND macro instruction locates a member of the data set for subsequent processing; the STOW macro instruction adds, deletes, replaces, or changes a member name in the directory. To use these macro instructions, you must specify DSORG=PO or POU in the DCB macro instruction. Before issuing a FIND, BLDL, or STOW macro instruction, you must check all preceding input/output operations for completion.

BLDL — Construct a Directory Entry List

The BLDL macro instruction is used to place directory information in main storage. The data is placed in a build list, which you construct before the BLDL macro instruction is issued. The format of the list is similar to that of the directory. For each member name in the list, the system supplies the address of the member and any additional information contained in the directory entry. Note that if there is more than one member name in the list, the member names must be in collating sequence regardless of whether the members are from the same library or from different libraries.

You can optimize retrieval time by directing a subsequent FIND macro instruction to the build list rather than the directory to locate the member to be processed.

The build list, as shown in Figure 44, must be preceded by a 4–byte list description that indicates the number of entries in the list and the length of each entry (12 to 76 bytes). The first 8 bytes of each entry contain the member name or alias. The next 6 bytes must be available to contain the starting address of the member plus some control data. If there is no user data entry, only the TTR and C fields are required. If additional information is to be supplied from the directory, up to 62 bytes can be reserved.



Programmer Supplies:

FF	Number of member entries in list.
LL	Even number giving byte length of each entry (minimum of 12).
Member name	Eight bytes, left-justified.

BLDL Supplies:

TTR	Member starting location.
K	If only data set = 0. If concatenation = number. Not required if no user data.
Z	Source of directory entry. Private library = 0. Link library = 1. Job or step library = 2. Not required if no user data.
C	Same C field from directory. Gives number of user data halfwords.
User data	As much as will fit in entry.

Figure 44. Build List Format

FIND — Position to a Member

To determine the starting address of a specific member, you must issue a FIND macro instruction. The system places the correct address in the data control block so that a subsequent input or output operation begins processing at that point.

There are two ways you can direct the system to the right member when you use the FIND macro instruction: specify the address of an area containing the name of the member or specify the address of the TTR field of the entry in a build list you have created by using the BLDL macro instruction. In the first case, the system searches the directory of the data set for the relative track address; in the second case, no search is required because the relative track address is in the build list entry.

If you want to process only one member, you can process it as a sequential data set (DSORG=PS) using either BSAM or QSAM. You indicate the name of the member you want to process and the name of the partitioned data set in the DSNAME parameter of the DD statement. When you open the data set, the system places the starting address in the data control block so that a subsequent GET or READ macro instruction begins processing at that point. You cannot use the FIND, BLDL, or STOW macro instructions when you are processing one member as a sequential data set.

STOW — Alter a Directory Entry

Unless you are adding members to a partitioned data set one at a time, you must issue a STOW macro instruction to enter the member name in the directory. When you add

a single member, the STOW operation is performed automatically when the data set is closed.

You can also use the STOW macro instruction to delete, replace, or change a member name in the directory, as well as to store additional information with the directory entry. Since an alias can also be stored in the directory in the same way, you should be consistent in altering all names associated with a given member. For example, if you replace a member, you must delete related aliases or change them so that they point to the new member. If you use STOW to change user data in the directory entry, you must also move the TTR of the member into the DCBRELAD.

If you do not use the STOW macro instruction before closing a partitioned data set that you have written, your CLOSE request causes the system to issue a STOW macro instruction. If you specify DISP=MOD, the system issues a STOW macro instruction with the replace option, causing replacement of an entry in the directory. If you specify DISP=NEW or DISP=OLD and the member does not exist, the system issues a STOW macro instruction with the add option, causing addition of an entry to the directory. If you specify DISP=OLD and the member already exists, the system issues a message to that effect.

Creating a Partitioned Data Set

If you have no need to add entries to the directory, that is, the STOW and BLDL macro instructions will not be used, you can create a new data set and write the first member as follows (see Figure 45):

- Code DSORG=PS or PSU in the DCB macro instruction.
- Indicate in the DD statement that the data is to be stored as a member of a new partitioned data set, that is, DSNAME=name (membername) and DISP=NEW.
- Request space for the member and the directory in the DD statement.
- Process the member with an OPEN macro instruction, a series of PUT or WRITE macro instructions, and then a CLOSE macro instruction. A STOW macro instruction is issued automatically when the data set is closed.

As a result of these steps, the data set and its directory are created, the records of the member are written, and a 12–byte entry is made in the directory.

```
//PDSDD    DD        ---,DSNAME=MASTFILE(MEMBERK),SPACE=(TRK,(100,5,7)),          C
              DISP=(NEW,KEEP)
OUTDCB     DCB       --,DSORG=PS,DDNAME=PDSDD,---
           ...
           OPEN      (OUTDCB,(OUTPUT))
           PUT[or WRITE]
           ..
           CLOSE               (OUTDCB)          Automatic Stow
```

Figure 45. Creating One Member of a Partitioned Data Set

To add additional members to the data set, follow the same procedure. However, a separate DD statement (with the space request omitted) is required for each member. The disposition should be specified as modify, DISP=MOD. The data set must be closed and reopened each time a new member is specified.

To take full advantage of the STOW macro instruction, and thus the BLDL and FIND macro instructions in future processing, you can provide additional information with each directory entry. You do this by using the basic access technique, which also allows you to process more than one member without closing and reopening the data set, as follows (see Figure 46):

- Request space in the DD statement for the members and the directory.
- Define DSORG=PO or POU in the DCB macro instruction.
- Use WRITE and CHECK to write and check the member records.
- Use NOTE to note the location of any note list written within the member, if there is a note list.
- When all the member records have been written, issue a STOW macro instruction to enter the member name, its location pointer, and any additional data in the directory.
- Continue to write, check, note, and stow until all the members of the data set and the directory entries have been written.

```
//PDSDD    DD        --,DSNAME=MASTFILE,SPACE=(TRK,(100,5,7)),DISP=MOD
OUTDCB     DCB       --,DSORG=PO,DDNAME=PDSDD,--
           OPEN      (OUTDCB,(OUTPUT))
           WRITE     **         Write and check first record of member.
           CHECK                The system will supply the relative
*                               track address for the directory entry.
           WRITE                Write and check remaining records of
           CHECK                number.
*
           NOTE                 If you are dividing the member into
           ST                   subgroups, note the location of the first
*                               record in subgroup, storing pointer
*                               in note list.
           WRITE                Write note list at end of member.
           CHECK
           NOTE                 Note location of note list, storing
           ST                   pointer in list for STOW.
           STOW                 Enter information in directory for
*                               this member after all records and note
*                               lists are written.
Repeat from ** for each additional member
           CLOSE(OUTDCB)
```

Figure 46. Creating Members of a Partitioned Data Set Using STOW

Retrieving a Member of a Partitioned Data Set

To retrieve a specific member from a partitioned data set, either the basic or queued access technique can be used as follows (see Figure 47):

- Code DSORG=PS or PSU in the DCB macro instruction.
- Indicate in the DD statement that the data is a member of an existing partitioned data set by coding DSNAME=name(membername) and DISP=OLD.
- Process the member with an OPEN macro instruction, a series of GET and READ macro instructions, and then a CLOSE macro instruction.

When your program is executed, the directory is searched automatically and the location of the member is placed in the DCB.

```
//PDSDD    DD     --,DSNAME=MASTFILE(MEMBERK),DISP=OLD
INDCB      DCB    --,DSORG=PS,DDNAME=PDSDD,--
           OPEN   (INDCB)                          Automatic Find
           GET (or READ)
           CLOSE  (INDCB)
```

Figure 47. Retrieving One Member of a Partitioned Data Set

```
//PDSDD    DD          --,DSNAME=MASTFILE,DISP=OLD
INDCB      DCB         --,DSORG=PO,DDNAME=PDSDD,--
           OPEN        (INDCB)
           BLDL                  Build a list of selected member names
                                 in main storage.
           FIND (or POINT)
/*
           READ                  *Read note list.
           CHECK
           POINT                 Locate subgroup by using note list.
           READ
           CHECK                 Read member records.
Repeat from * for each additional member.
           CLOSE       (INDCB)
```

Figure 48. Retrieving Several Members of a Partitioned Data Set Using BLDL, FIND, and POINT

To process several members without closing and reopening, or to take advantage of additional data in the directory, the following technique should be used (see Figure 48):

- Code DSORG=PO or POU in the DCB macro instruction.
- Build a list (BLDL) of needed member entries from the directory.
- Indicate in the DD statement the data set name of the partitioned data set by coding DSNAME=name and DISP=OLD.
- Use the FIND or POINT macro instruction to prepare for reading the member records.

- The records may be read from the beginning of the member, or a note list may be read first, to obtain additional locations that point to subcategories within the member.
- Read (and check) the records until all those required have been processed.
- Point to additional categories, if required, and read the records.
- Repeat this procedure for each member to be retrieved.

Updating a Member of a Partitioned Data Set

A member of a partitioned data set can be updated in place, or can be deleted and rewritten as a new member.

Updating in Place

When you update in place, you read records, process them, and write them back to their original positions without destroying the remaining records on the track. The following rules apply:

- You must specify the update option (UPDAT) in the OPEN macro instruction. To perform the update, you can use only the READ, WRITE, CHECK, NOTE, POINT, FIND, and BLDL macro instructions.
- You cannot use chained scheduling.
- You cannot delete any record or change its length; you cannot add new records.

A record must be retrieved by a READ macro instruction before it can be updated by a WRITE macro instruction. Both macro instructions must be execute forms that refer to the same DECB; the DECB must be provided by a list form. (The execute and list forms of the READ and WRITE macro instructions are described in *OS Data Management Macro Instructions.*)

Updating With Overlapped Operations: To overlap input/output and CPU activity, you can start several read or write operations before checking the first for completion. You cannot overlap read and write operations, however, as operations of one type must be checked for completion before operations of the other type are started or resumed. Note that each concurrent read or write operation requires a separate channel program and a separate DECB. If a single DECB were used for successive read operations, only the last record read could be updated.

In Figure 49, overlap is achieved by having a read or write request outstanding while each record is being processed. Note the use of execute–form and list–form macro instructions, identified by the operands MF=E and MF=L.

Rewriting a Member

There is no actual update option that can be used to add or extend records in a partitioned data set. If you want to extend or add a record within a member, you must rewrite the complete member in another area of the data set. Since space is allocated when the data set is created, there is no need to request additional space. Note, however, that a partitioned data set must be contained on one volume. If sufficient space has not been allocated, the data set must be reorganized by the IEBCOPY utility program.

```
//PDSDD  DD       DSNAME=MASTFILE(MEMBERK),DISP=OLD,---

UPDATDCB DCB     DSORG=PS,DDNAME=PDSDD,MACRF=(R,W),NCP=2,EODAD=FINISH
         READ    DECBA,SF,UPDATDCB,AREAA,MF=L          Define DECBA
         READ    DECBB,SF,UPDATDCB,AREAB,MF=L          Define DECBB
AREAA    DS      ---                                   Define buffers
AREAB    DS      ---
         ...
         OPEN    (UPDATDCB,UPDAT)                      Open for update
         LA      2,DECBA                               Load DECB addresses
         LA      3,DECBB
READRECD READ    (2),SF,MF=E                           Read a record
NEXTRECD READ    (3),SF,MF=E                           Read the next record
         CHECK   (2)                                   Check previous read operation
         (If update is required, branch to R2UPDATE)

         LR      4,3                                   If no update is required,
         LR      3,2                                   switch DECB addresses in
         LR      2,4                                   registers 2 and 3
         B       NEXTRECD                              and loop
```

In the following statements, "R2" and "R3" refer to the records that were read using the DECBs whose addresses are in registers 2 and 3, respectively. Either register may point to either DECBA or DECBB.

```
R2UPDATE CALL    UPDATE,((2))                          Call routine to update R2
         CHECK   (3)                                   Check read for next record (R3)
         WRITE   (2),SF,MF=E                           Write updated R2
         (If R3 requires an update, branch to R3UPDATE)
         CHECK   (2)                                   If R3 requires no update, check
         B       READRECD                              write for R2 and loop
R3UPDATE CALL    UPDATE,((3))                          Call routine to update R3
         WRITE   (3),SF,MF=E                           Write updated R3
         CHECK   (2)                                   Check write for R2
         CHECK   (3)                                   Check write for R3
         B       READRECD                              Loop
FINISH   CLOSE   (UPDATDCB)                            End-of-Data exit routine
         ...
```

Figure 49. Updating a Member of a Partitioned Data Set

When you rewrite the member, you must provide two DCBs, one for input and one for output. Both DCB macro instructions can refer to the same data set, that is, only one DD statement is required.

You can reflect the change in location of the member either automatically, by indicating a disposition of OLD, or by using the STOW macro instruction. Although the old member is, in effect, deleted, its space cannot be reused until the data set is reorganized.

Processing an Indexed Sequential Data Set

The organization of an indexed sequential data set allows you a great deal of flexibility in the operations you can perform. The data set can be read or written sequentially, individual records can be processed in any order, records can be deleted, and new records can be added. The system automatically locates the proper position in the data set for new records and makes any necessary adjustments when records are deleted.

The queued access technique must be used to create an indexed sequential data set. It can also be used to sequentially process or update the data set and to add records to

the end of the data set. The basic access technique can be used to insert new records between records already in the data set and to update the data set directly.

Indexed Sequential Data Set Organization

The records in an indexed sequential data set are arranged according to collating sequence by a *key field* in each record. Each block of records is preceded by a key field that corresponds to the key of the last record in the block.

An indexed sequential data set resides on direct–access storage devices and can occupy up to three different areas:

- *Prime Area* — This area, also called the prime data area, contains data records and related track indexes. It exists for all indexed sequential data sets.
- *Overflow Area* — This area contains records that overflow from the prime area when new data records are added. It is optional.
- *Index Area* — This area contains master and cylinder indexes associated with the data set. It exists for a data set that has a prime area occupying more than one cylinder.

The indexes of an indexed sequential data set are analogous to the card catalog in a library. For example, if the library user knows the name of the book or the author, he can look in the card catalog and obtain a catalog number that will enable him to locate the book in the book files. He would then go to the shelves and proceed through rows until he found the shelf containing the book. Usually each row contains a sign to indicate the beginning and ending numbers of all books in that particular row. Thus, as he proceeded through the rows, he would compare the catalog number obtained from the index with the numbers posted on each row. Upon locating the proper row, he would then search that row for the shelf that contained the book. Then he would look at the individual book numbers on that shelf until he found the particular book.

ISAM uses the indexes in much the same way to locate records in an indexed sequential data set.

As the records are written in the prime area of the data set, the system accounts for the records contained on each track in a *track index area.* Each entry in the track index identifies the key of the last record on each track. There is a track index for each cylinder in the data set. If more than one cylinder is used, the system develops a higher–level index called a *cylinder index.* Each entry in the cylinder index identifies the key of the last record in the cylinder. To increase the speed of searching the cylinder index, you can request that a *master index* be developed for a specified number of cylinders, as shown in Figure 50.

Rather than reorganize the whole data set when records are added, you can request that space be allocated for additional records in an overflow area.

Prime Area

Records are written in the prime area when the data set is created or updated. The portion of Figure 50 labeled Cylinder 1 illustrates the initial structure of the prime area. Although the prime area can extend across several noncontiguous areas of the volume, all the records are written in key sequence. Each record must contain a key; the system automatically writes the key of the highest record before each block.

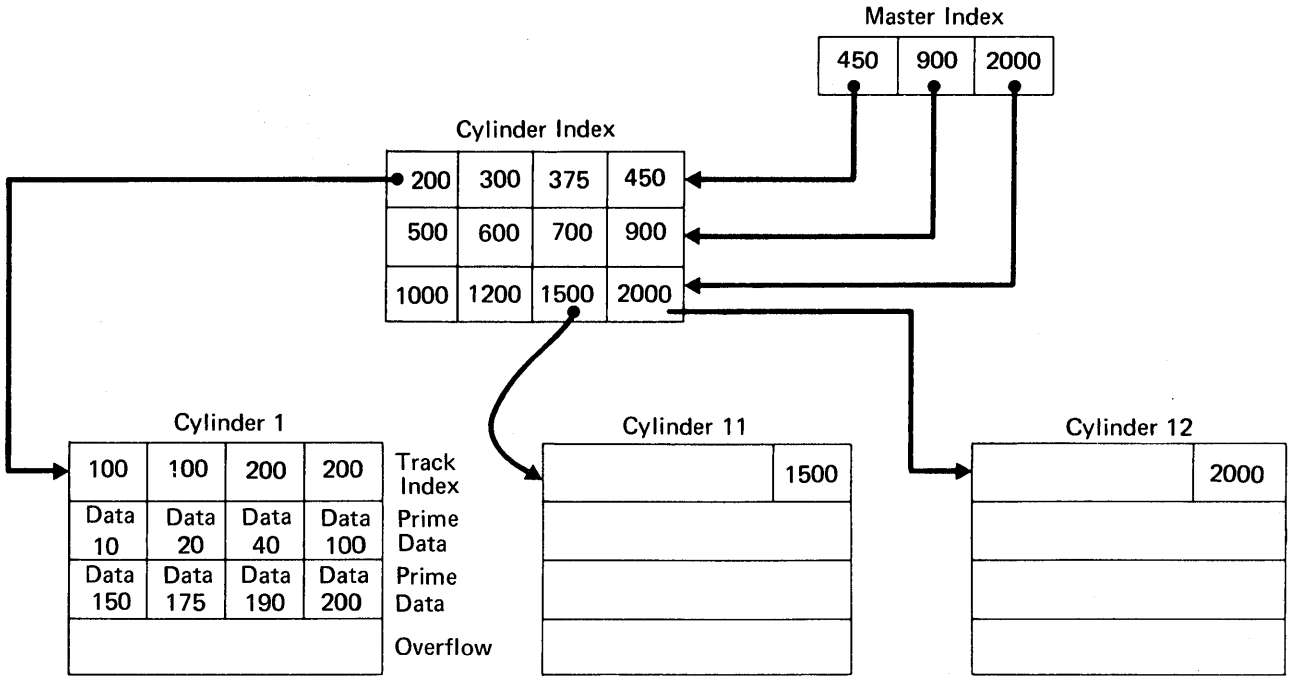


Figure 50. Indexed Sequential Data Set Organization

When the ABSTR option of the SPACE parameter of the DD statement is used to generate a multivolume prime area, the VTOC of the second volume and on all succeeding volumes must be contained within cylinder 0 of the volume.

Index Areas

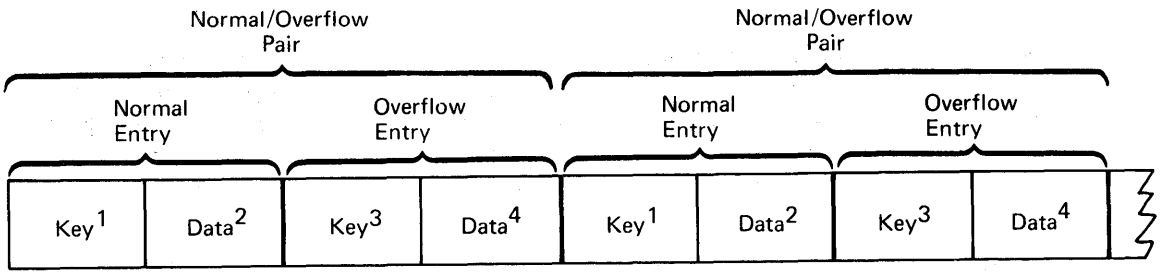
The operating system generates track and cylinder indexes automatically. Up to three levels of master indexes are created if requested.

Track Index: This is the lowest level of index and is always present. There is one track index for each cylinder in the prime area; it is written on the first track(s) of the cylinder that it indexes.

The index consists of a series of paired entries, that is, of a normal entry and an overflow entry for each prime track. For fixed–length records, each normal entry (and also DCBFIRSH) points to either record 0 or the first prime record on a shared track. For variable–length records, the normal entry contains the key of the highest record on the track and the address of the last record on the track. The overflow entry is originally the same as the normal entry. (This is why 100 appears twice on the track index for cylinder 1 in Figure 50.) The overflow entry is changed when records are added to the data set. Then the overflow entry contains the key of the highest overflow record and the address of the lowest overflow record logically associated with the track. Figure 51 shows the format of a track index.

If all the tracks allocated for the prime data area are not used, the index entries for the unused ones are flagged as inactive. The last entry of each track index is a dummy entry indicating the end of the index. When fixed–length record format has been

specified, the remainder of the last track used for a track index contains prime data records if there is room for them.



[1]Normal key = key of the highest record on the prime data track

[2]Normal data = address of the prime data track

[3]Overflow key = key of the highest overflow record logically associated with the prime data track

[4]Overflow data = address of the lowest overflow record logically associated with the prime data track

Notes:

- If there are no overflow records, overflow key and data entries are the same as normal key and data entries.
- This figure is a logical representation only; that is, it makes no attempt to show the physical size of track index entries.

Figure 51. Format of Track Index Entries

Each index entry has the same format. It is an unblocked, fixed–length record consisting of a count, a key, and a data area. The length of the key corresponds to the length of the key area in the record to which it points. The data area is always 10 bytes long. It contains the full address of the track or record to which the index points, as well as the level of the index and the entry type.

Cylinder Index: For every track index created, the system generates a cylinder index entry. There is one cylinder index for a data set, each entry of which points to a track index. Since there is one track index per cylinder, there is one cylinder index entry for each cylinder in the prime data area, except in the case of a 1–cylinder prime area. As with track indexes, inactive entries are created for any unused cylinders in the prime data area.

Master Index: As an optional feature, the operating system creates, at your request, a master index. Each entry in the master index points to a track of the cylinder index. This avoids a serial search through a large cylinder index.

You can specify the number of entries to be included in each master index. For example, if you indicate that you want a master index created for every 3 tracks of cylinder index entries, a master index is created if the cylinder index exceeds 3 tracks. A higher–level master index is created if the first level master index exceeds 3 tracks. This procedure continues up to three levels of master indexes.

Overflow Areas

As records are added to an indexed sequential data set, space is required to contain those records that will not fit on the prime data track on which they belong. You can request that a number of tracks be set aside as a *cylinder overflow area* to contain

overflows from prime tracks in each cylinder. An advantage of using cylinder overflow areas is a reduction of search time required to locate overflow records. A disadvantage is that there will be unused space if the additions are unevenly distributed throughout the data set.

Instead of, or in addition to, cylinder overflow areas, you can request an *independent overflow area.* Overflow from anywhere in the prime data area is placed in a specified number of cylinders reserved solely for overflow records. An advantage of having an independent overflow area is a reduction in unused space reserved for overflow. A disadvantage is the increased search time required to locate overflow records in an independent area.

If you request both cylinder overflow and independent overflow, the cylinder overflow area is used first. It is a good practice to request cylinder overflow areas large enough to contain a reasonable number of additional records and an independent overflow area to be used as the cylinder overflow areas are filled.

Adding Records to an Indexed Sequential Data Set

Either the queued access technique or the basic access technique may be used to add records to an indexed sequential data set. A record to be inserted between records already in the data set must be inserted by the basic access method using WRITE KN (key new). Records added to the end of a data set, that is, records with successively higher keys, may be added to the prime data area or the overflow area by the basic access method using WRITE KN, or they may be added to the prime data area by the queued access technique using the PUT macro instruction.

Inserting New Records into an Existing Indexed Sequential Data Set

As you add records to an indexed sequential data set, the system inserts each record in its proper sequence according to the record key. The remaining records on the track are then moved up one position each. If the last record does not fit on the track, it is written in the first available location in the overflow area. A 10–byte *link field* is added to the record put in the overflow area to connect it logically to the correct track. The proper adjustments are made to the track index entries. This procedure is illustrated in Figure 52.

Subsequent additions are written either on the prime track or as part of the *overflow chain* from that track. If the addition belongs after the last prime record on a track but before a previous overflow record from that track, it is written in the first available location in the overflow area. Its link field contains the address of the next record in the chain.

Adding New Records to the End of an Indexed Sequential Data Set

Records added to the end of a data set, that is, records with successively higher keys, may be added by the basic access method using WRITE KN (key new), or by the queued access method using the PUT macro instruction (resume load). In either case records may be added to the prime data area.

When you use the WRITE KN macro instruction, the record being added is placed in the prime data area only if there is room for it on the prime data track containing the record with the highest key currently in the data set. If there is not sufficient room on that track, the record is placed in the overflow area and linked to that prime track even though additional prime data tracks originally allocated have not been filled.

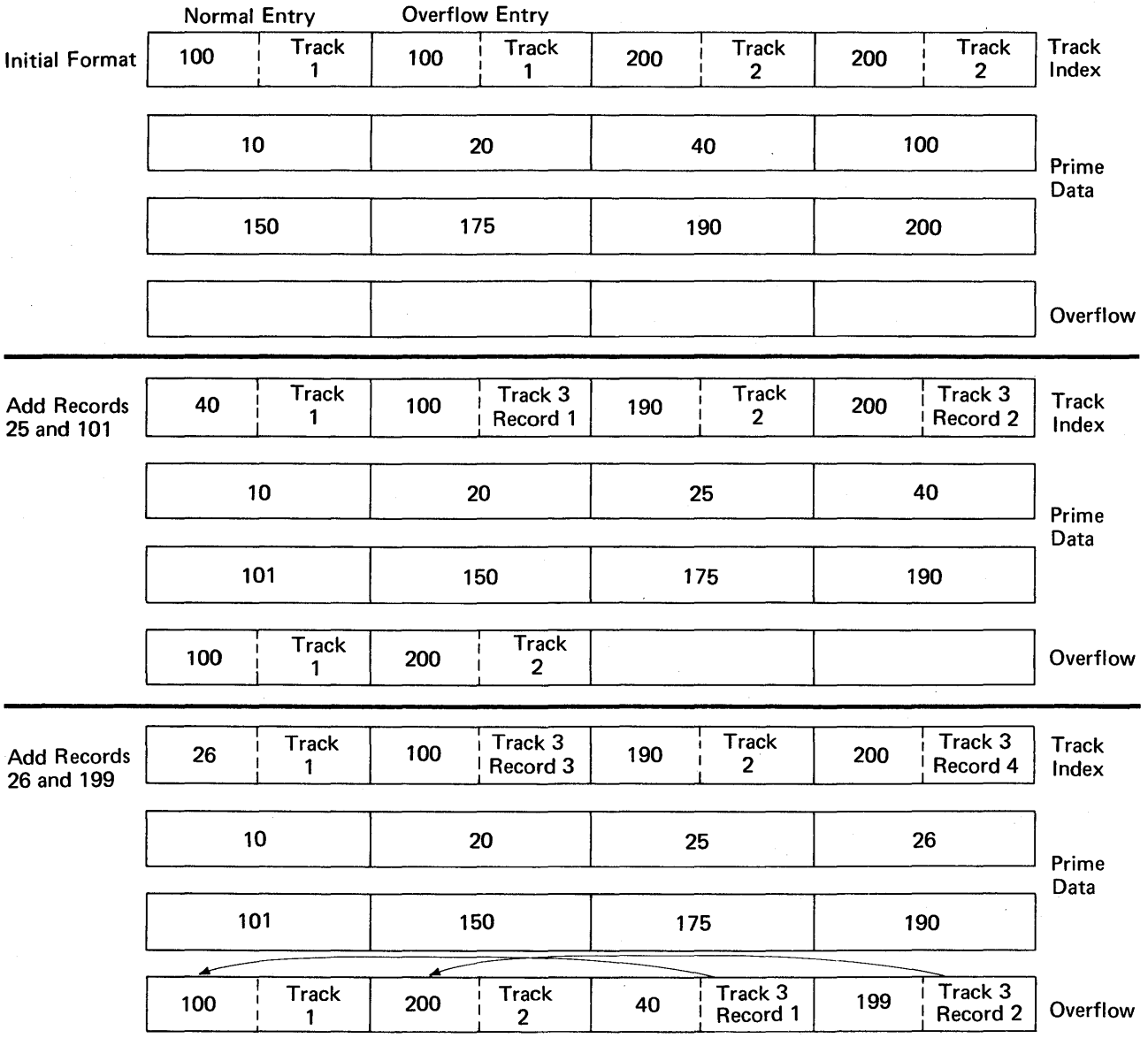


Figure 52. Adding Records to an Indexed Sequential Data Set

When you use the PUT macro instruction (resume load), records are added to the prime data area until the space originally allocated is filled. Once this allocated prime area is filled, you can add records to the data set using WRITE KN, in which case they will be placed in the overflow area. Resume load is discussed in more detail later under "Creating an Indexed Sequential Data Set."

In order to add records with successively higher keys using the PUT macro instruction (resume load):

- The key of any record to be added must be higher than the highest key currently in the data set.

- The DD statement must specify DISP=MOD.
- The data set must have been successfully closed when it was created or when records were previously added using the PUT macro instruction.

You may continue to add fixed–length records in this manner until the original space allocated for prime data is exhausted.

When you add records to an indexed sequential data set using the PUT macro instruction (resume load), new entries are also made in the indexes. During resume load on a data set with a partially filled track and/or a partially filled cylinder, the track index entry and/or the cylinder index entry is overlaid when the track or cylinder is filled. If resume load abnormally terminates after these index entries have been overlaid, a subsequent resume load will get a sequence check when adding a key that is higher than the highest key at the last successful CLOSE but lower than the key in the overlaid index entry. When the SYNAD exit is taken for a sequence check, register 0 contains the address of the highest key of the data set.

Maintaining an Indexed Sequential Data Set

An indexed sequential data set must be reorganized occasionally for two reasons:

- The overflow area will eventually be filled.
- Additions increase the time required to locate records directly.

The frequency of reorganization depends on the activity of the data set and on your timing and storage requirements. There are two ways you can accomplish reorganization:

- You can reorganize the data set in two passes by writing it sequentially into another area of direct–access storage or magnetic tape and then recreating it in the original area.
- You can reorganize the data set in one pass by writing it directly into another area of direct–access storage. In this case, the area occupied by the original data set cannot be used by the reorganized data set.

The operating system maintains statistics that are pertinent to reorganization. The statistics, written on the direct–access volume and available in the DCB for checking, include the number of cylinder overflow areas, the number of unused tracks in the independent overflow area, and the number of references to overflow records other than the first. They appear in the RORG1, RORG2, and RORG3 fields of the DCB.

If you indicate when creating or updating the data set that you want to be able to flag records for deletion during updating, you can set the delete code (the first byte of a fixed–length record or the fifth byte of a variable–length record) to X'FF'. If a flagged record is forced off its prime track during a subsequent update, it will not be rewritten in the overflow area, as shown in Figure 53, unless it has the highest key on that cylinder. Similarly, when you process sequentially, flagged records are not retrieved for processing. During direct processing, flagged records are retrieved like any other records, and you should check them for the delete code.

Note that to use the delete option, RKP must be greater than 0 for fixed–length records and greater than 4 for variable–length records.

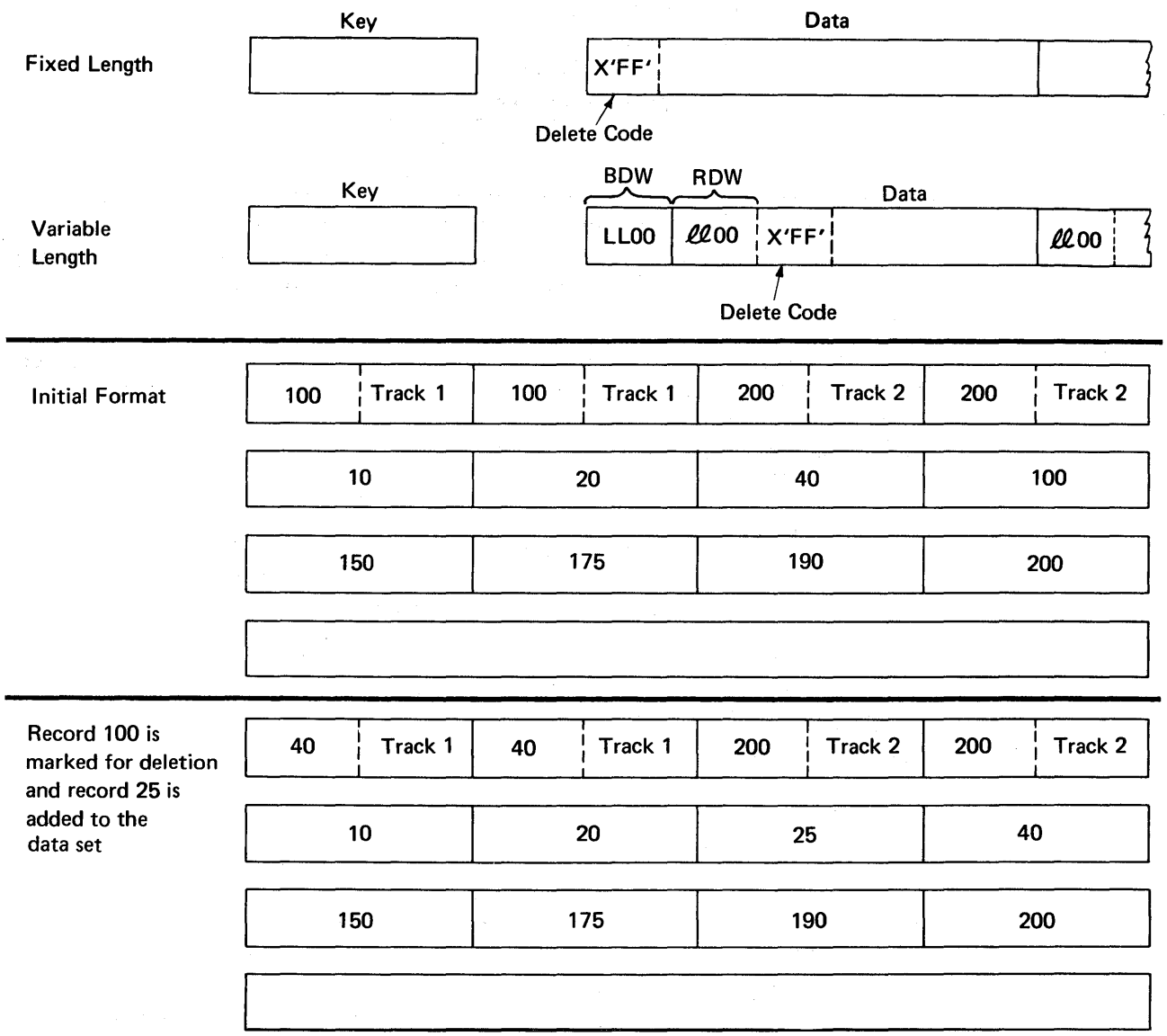
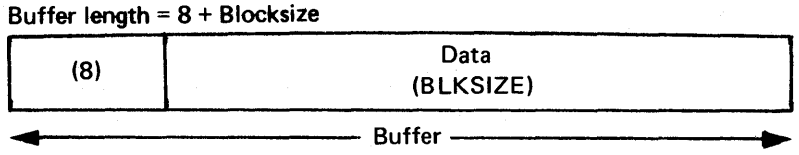


Figure 53. Deleting Records from an Indexed Sequential Data Set

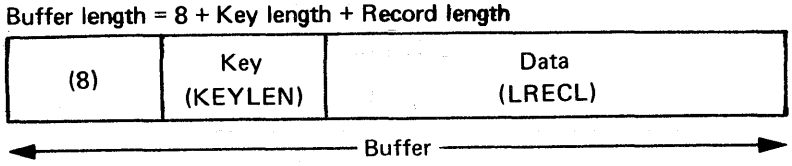
Indexed Sequential Buffer and Work Area Requirements

The only case in which you will ever have to compute the buffer length (BUFL) requirements for your program is when you use the BUILD or GETPOOL macro instruction to construct the buffer area. If you are creating an indexed sequential data set (using the PUT macro instruction), each buffer must be 8 bytes longer than the blocksize to allow for the hardware count field, that is:



One exception to this formula arises when you are dealing with an unblocked format–F record whose key field precedes the data field; its relative key position is 0 (RKP=0). In that case the key length must also be added, that is:

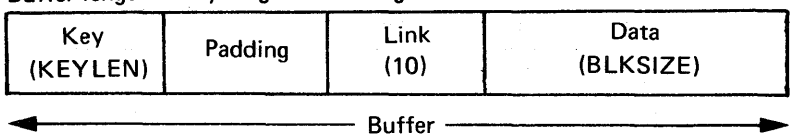
Buffer length = 8 + Key length + Record length



The buffer requirements for using the queued access technique to read or update (using the GET or PUTX macro instruction) an indexed sequential data set are discussed below.

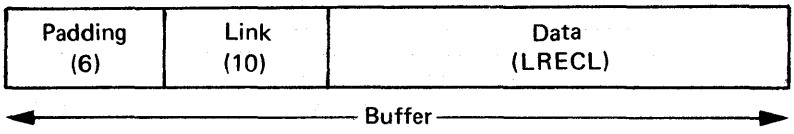
For fixed–length unblocked records when both the key and data are to be read and for variable–length unblocked records, padding is added so that the data will be on a doubleword boundary, that is:

Buffer length = Key length + Padding + 10 + Blocksize



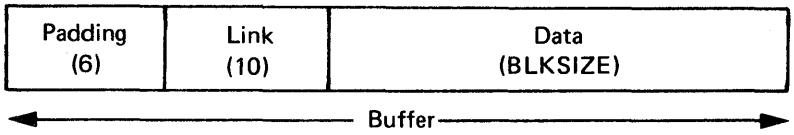
For fixed–length unblocked records when only data is to be read:

Buffer length = 16 + LRECL



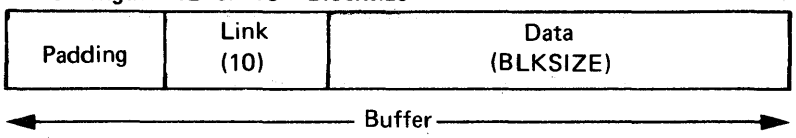
For fixed–length blocked records:

Buffer length = 16 + Blocksize



For variable–length blocked records, padding is 2 if the buffer starts on a fullword boundary that is not also a doubleword boundary or 6 if the buffer starts on a doubleword boundary, that is:

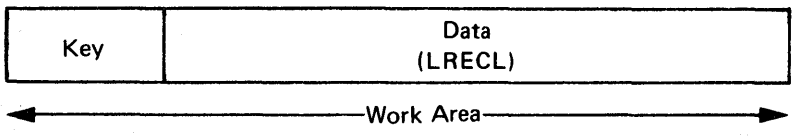
Buffer length = 12 or 16 + Blocksize



If you are using the input data set with fixed–length, unblocked records as a basis for creating a new data set, a work area is required.

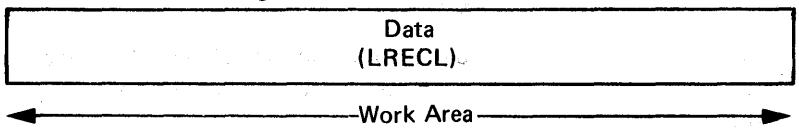
The size of the work area is given by:

Work area = Key length + Record length



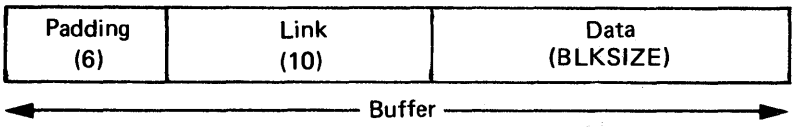
If you are reading only the data portion of fixed–length unblocked records or variable–length records, the work area is the same size as the record, that is:

Work area = Record length



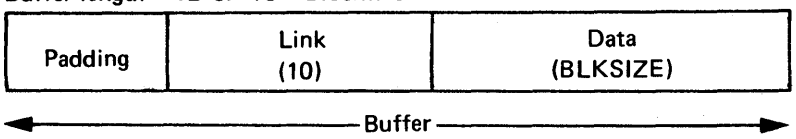
When you use the basic access technique to update records in an indexed sequential data set, the key length field need not be considered in determining your buffer requirements. The area for fixed–length records must be:

Buffer length = 16 + Blocksize



For variable–length records, padding is 2 if the buffer starts on a fullword boundary that is not also a doubleword boundary or 6 if a buffer starts on a doubleword boundary. Thus, the area must be:

Buffer length = 12 or 16 + Blocksize



You can speed up the process of adding fixed–length or variable–length records to a data set by using the MSWA parameter of the DCB macro instruction to provide a special work area for the operating system. The size of the work area (SMSW parameter in the DCB) must be large enough to contain a full track of data, the count fields of each block, and the work space for inserting the new record.

The size of the work area needed varies according to the record format and the device type. You can calculate it during execution using device–dependent information obtained with the DEVTYPE macro instruction and data set information from the DSCB obtained with the OBTAIN macro instruction. The DEVTYPE and OBTAIN macro instructions are discussed in *OS Data Management for System Programmers.*

Note that you can use the DEVTYPE macro instruction only if the index and prime areas are on devices of the same type or if the index area is on a device with a larger track capacity than that of the device containing the prime area. If you are not trying to maintain device independence, you may precalculate the size of the work area needed and specify it in the SMSW field of the DCB macro instruction. The maximum value for SMSW is 65,535.

For calculating the size of the work area, refer to the storage device capacities shown in Figure 61 under "Estimating Space Requirements" and the device overhead formulas given in the same section.

For fixed–length blocked records, SMSW is calculated as follows:

```
SMSW = HIRPD(BLKSIZE + 8) + LRECL + KEYLEN
```

The formula for fixed–length unblocked records is

```
SMSW = HIRPD(KEYLEN + LRECL + 8) + 2
```

The value for HIRPD is in the index (format 2) DSCB. *OS System Control Blocks* shows the exact location of this field in the index DSCB. If you don't use the MSWA and SMSW parameters, the control program supplies a work area using the formula BLKSIZE + LRECL + KEYLEN.

For variable–length records, SMSW may be calculated by one of two methods. The first method may lead to faster processing although it may require more main storage than the second method.

The first method is as follows:

$$SMSW = HIRPD(BLKSIZE + 8) + LRECL + KEYLEN + 10$$

The second method is as follows:

$$SMSW = \left(\frac{\text{Track Capacity} - Bn + 1}{Bi}\right)(BLKSIZE) + 8(HIRPD) + LRECL + KEYLEN + 10 + (REM - N - KEYLEN)$$

In all of the above formulas, the terms BLKSIZE, LRECL, KEYLEN, and SMSW are the same as the parameters in the DCB macro instruction. REM is the remainder of the division operation in the formula and N is the first constant in the Bi formulas described in Figure 62. (REM–N–KEYLEN) is added only if it is positive. The second method yields a minimum value for SMSW. Therefore, the first method is valid only if its application results in a value higher than the value that would be derived from the second method. If neither MSWA nor SMSW is specified, the control program supplies the work area for variable–length records, using the second method to calculate the size.

Another technique to increase the speed of processing is to provide space in main storage for the highest–level index. To specify the address of this area, use the MSHI operand of the DCB. When the address of this area is specified, you must also specify its size, which you can do by using the SMSI operand of the DCB. The maximum value for SMSI is 65,535. If you do not use this technique, the index on the volume must be searched.

The size of the storage area (SMSI parameter) varies. To allocate that space during execution, you can find the size of the high–level index in the DCBNCRHI field of the DCB during your DCB exit routine or after the data set is open. Use the DCBD macro instruction to gain access to the DCBNCRHI field (see "Modifying the Data Control Block" in Part 1). You can also find the size of the high–level index in the DS2NOBYT field of the index (format 2) DSCB, but you must use the utility program IEHLIST to print the information in the DSCB. You can calculate the size of the storage area required for the high–level index by using the formula

$$SMSI = \left(\begin{matrix}\text{Number of Tracks} \\ \text{in High–Level Index}\end{matrix}\right)\left(\begin{matrix}\text{Number of Entries} \\ \text{per Track}\end{matrix}\right)(\text{Key Length} + 10)$$

The formula for calculating the number of tracks in the high–level index is in the section "Calculating Space Requirements for an Indexed Sequential Data Set" in Part 3. When a data set is shared and has the DCB integrity feature (DISP=SHR), the high-level index in storage is not updated when DCB fields are changed.

Controlling an Indexed Sequential Data Set Device

An indexed sequential data set is processed sequentially or directly. Direct processing is accomplished by the basic access technique. Because you provide the key for the record you want read or written, all device control is handled automatically by the

system. If you are processing the data set sequentially, using the queued access technique, the device is automatically positioned at the beginning of the data set.

In some cases, you may wish to process only a section or several separate sections of the data set. You do this by using the SETL macro instruction, which directs the system to begin sequential retrieval at the record having a specific key. The processing of succeeding records is the same as for normal sequential processing, except that you must recognize when the last desired record has been processed. At this point, issue the ESETL macro instruction to terminate sequential processing. You can then begin processing at another point in the data set.

SETL — Specify Start of Sequential Retrieval

The SETL macro instruction enables you to retrieve records starting at the beginning of an indexed sequential data set or at any point in the data set. Processing that is to start at a point other than the beginning can be requested in the form of a record key, a key prefix, or an actual address of a prime data record.

Use of a key prefix is useful because you don't have to know the whole key of the first record to be processed. Any number of key characters can be used in the key prefix. Key characters to the right of the key prefix should be represented by binary 0s.

To use actual addresses, you must keep an account of where the records were written when the data set was created. The device address of the block containing the record just processed by a PUT–move macro instruction is available in the 8–byte data control block field DCBLPDA. For blocked records the address is the same for each record in the block.

ESETL — End Sequential Retrieval

The ESETL macro instruction directs the system to stop retrieving records from an indexed sequential data set. A new scan limit can then be set, or processing terminated. An end–of–data–set indication automatically terminates retrieval. An ESETL macro instruction must be executed before another SETL macro instruction (described above) using the same DCB is executed.

Creating an Indexed Sequential Data Set

You can create an indexed sequential data set in one step or in several steps. You can create the data set either by writing all records in a single step or by writing one group of records in one step and writing additional groups of records in subsequent steps. Writing records in subsequent steps is *resume loading*. When using either one step or several steps, you must present the records for writing in ascending order by key.

To create an indexed sequential data set by the one–step method, you should proceed as follows:

- Code DSORG=IS or ISU and MACRF=PM or PL in the DCB macro instruction.
- Specify in the DD statement the DCB attributes DSORG=IS or ISU, record length (LRECL), blocksize (BLKSIZE), record format (RECFM), key length (KEYLEN), relative key position (RKP), options required (OPTCD), cylinder overflow (CYLOFL), and the number of tracks for a master index (NTM).

Specify space requirements with the SPACE parameter. To reuse previously allocated space, omit the SPACE parameter and code DISP=(OLD,[KEEP]).

- Open the data set for output.
- Use the PUT macro instruction to place all the records or blocks on the direct–access volume.
- Close the data set.

The records that compose a newly created data set must be presented for writing in ascending order by key. You can merge two or more input data sets. If you want a data set with no records (a null data set), you must write at least one record when you create the data set. You can subsequently delete this record to achieve the null data set.

If records are blocked, you should not write a 1–byte record with the hexadecimal value FF. This value is used for padding; if it occurs as the last record of a block, the record cannot be retrieved.

When creating an indexed sequential data set, a procedure called *loading,* you can improve performance by using the full–track–index–write option. You do this by specifying OPTCD=U in the DCB. This causes the operating system to accumulate track–index entries in main storage. Note that the full–track–index–write option can be used only for fixed–length records.

If you do not specify this option, the operating system writes each normal–overflow pair of entries for the track index after the associated prime data track has been written. If you specify this option, the operating system accumulates track–index entries in main storage until either there are enough entries to fill a track or end–of–data or end–of–cylinder is reached. Then the operating system writes these entries as a group, writing one group for each track of track index.

When you specify the full–track–index–write option, the track index entries are written as fixed–length unblocked records. If a large enough area of main storage is not available, the entries are written as they are created, that is, in normal–overflow pairs.

Once an indexed sequential data set has been created, its characteristics cannot be changed. However, for added flexibility, the system allows you to retrieve records using either the queued access technique with simple buffering, or the basic access technique with dynamic buffering.

Tape–to–Disk — Indexed Sequential Data Set: The example in Figure 54 shows the creation of an indexed sequential data set from an input tape containing 60–character records. The key by which the data set is organized is in positions 20–29. The output records will be an exact image of the input, except that the records will be blocked. One track per cylinder is to be reserved for cylinder overflow. Master indexes are to be built when the cylinder index exceeds six tracks. Reorganization information about the status of the cylinder overflow areas is to be maintained by the system. The delete option will be used during any future updating.

To create an indexed sequential data set in more than one step, create the first group of records using the one step method described above. •This first section must contain at least one data record. The remaining records can then be added to the end of the data set in subsequent steps using resume load. Each group to be added must contain records with successively higher keys. This method allows you to create the indexed sequential data set in several short time periods rather than in a single long one.

```
//INDEXDD  DD       DSNAME=SLATE.DICT(PRIME),DCB=(BLKSIZE=240,CYLOFL=1,          C
//                  DSORG=IS,OPTCD=MYLR,RECFM=FB,LRECL=60,NTM=6,RKP=19,          C
                    KEYLEN=10),UNIT=2311,SPACE=(CYL,25,,CONTIG),---
//INPUTDD  DD       ---
ISLOAD   START     0
         ...
         DCBD      DSORG=IS
ISLOAD   CSECT
         OPEN      (IPDATA,,ISDATA,(OUTPUT))
NEXTREC  GET       IPDATA                          Locate mode
         LR        0,1                             Address of record in register 1
         PUT       ISDATA,(0)                      Move mode
         B         NEXTREC
         ...
CHECKERR L         3,=A(ISDATA)                    Initialize base for errors
         USING     IHADCB,3
         TM        DCBEXCD1,X'04'
         BO        OPERR                           Uncorrectable error
         TM        DCBEXCD1,X'20'
         BO        NOSPACE                         Space not found
         TM        DCBEXCD2,X'80'
         BO        SEQCHK                          Record out of sequence
Rest of error checking
Error routine
End of job routine (EODAD FOR IPDATA)
IPDATA   DCB       ---
ISDATA   DCB       DDNAME=INDEXDD,DSORG=IS,MACRF=(PM),SYNAD=CHECKERR
```

Figure 54. Creating an Indexed Sequential Data Set

This method also allows you to provide limited recovery from uncorrectable output errors. When an uncorrectable output error is detected, do not attempt to continue processing or to close the data set. If you have provided a SYNAD routine, it should issue the ABEND macro instruction to terminate processing. If no SYNAD routine is provided, the control program will terminate your processing. If the error shows that space in which to add the record was not found, you must close the data set; issuing subsequent PUT macro instructions can cause unpredictable results. You should begin recovery at the record following the end of the data as of the last successful close. The rerun time is limited to that necessary to add the new records, rather than to that necessary to recreate the whole data set.

When you extend an indexed sequential data set with resume load, the disposition parameter of the DD statement must specify MOD. To ensure that the necessary control information is in the DSCB before attempting to add records, you should at least open and close the data set successfully on a version of the system that includes resume load. This need be done only if the data set was created on a previous version of the system. Records may be added to the data set by resume load until the space allocated for prime data in the first step has been filled.

During resume load on a data set with a partially filled track and/or a partially filled cylinder, the track index entry and/or the cylinder index entry is overlaid when the track or cylinder is filled. If resume load abnormally terminates after these index entries have been overlaid, a subsequent resume load will result in a sequence check when it adds a key that is higher than the highest key at the last successful CLOSE but lower than the key in the overlaid index entry. When the SYNAD exit is taken for a sequence check, register 0 contains the address of the high key of the data set.

Updating an Indexed Sequential Data Set

To sequentially retrieve and update records in an indexed sequential data set:

- Code DSORG=IS or ISU to agree with what you specified when you created the data set, and MACRF=GL, SK, or PU in the DCB macro instruction.
- Code a DD statement for retrieving the data set. The data set characteristics and options are as defined when the data set was created.
- Open the data set.
- Set the beginning of sequential retrieval (SETL).
- Retrieve records and process as required, marking records for deletion as required.
- Return records to the data set.
- Use ESETL to end sequential retrieval as required and reset the starting point.
- Close the data set to end all retrieval.

Sequential Updates — Indexed Sequential Data Set: Assume that, using the data set created in the previous example, you are to retrieve all records beginning with 915. Those records with a date (positions 13–16) before today's date are to be deleted. The date is in the standard form as returned by the system in response to the TIME macro instruction, that is, packed decimal 00yyddds. Overflow records can be logically deleted even though they cannot be physically deleted from the data set.

One way to solve this problem is shown in Figure 55.

Direct Retrieval and Update of an Indexed Sequential Data Set

By using the basic indexed sequential access method (BISAM) to process an indexed sequential data set, you can make direct references to the records in the data set for the purpose of:

- Direct retrieval of a record by its key
- Direct update of a record
- Direct insertion of new records

Because the operations are direct, there can be no anticipatory buffering. However, the system provides dynamic buffering each time a read request is made, if specified.

To ensure that the requested record is in main storage before you start processing, you must issue a WAIT or CHECK macro instruction. If you issue a WAIT macro instruction, you must test the exception code field of the DECB. If you issue a CHECK macro instruction, the system tests the exception code field in the DECB. If an error analysis routine has not been specified and a CHECK is issued, the program is abnormally terminated with a system completion code X'001'. In either case, if you wish to determine whether the record is an overflow record, you should test the exception code field of the DECB.

After you test the exception code field of the DECB, you need not set it to 0. If you have used a READ KU macro instruction and if you plan to use the same DECB again to rewrite the updated record using a WRITE K macro instruction, you should not set the field to 0. If you do, your record may not be rewritten properly.

```
//INDEXDD  DD       DSNAME=SLATE.DICT,---
ISRETR   START    0
         DCBD     DSORG=IS
ISRETR   CSECT
         ...
         USING    IHADCB,3
         LA       3,ISDATA
         OPEN     (ISDATA)
         SETL     ISDATA,KC,KEYADDR          Set scan limit
         TIME                                Today's date in register 1
         ST       1,TODAY
NEXTREC  GET      ISDATA                     Locate mode
         CLC      19(10,1),LIMIT
         BNL      ENDJOB
         CP       12(4,1),TODAY              Compare for old date
         BNL      NEXTREC
         MVI      0(1),X'FF'                 Flag old record for deletion
         PUTX     ISDATA                     Return delete record
         B        NEXTREC
TODAY    DS       F
KEYADDR  DC       C'915'                     Key prefix
         DC       XL7'0'                     Key padding
LIMIT    DC       C'916'
         DC       XL7'0'
         ...
CHECKERR
Test DCBEXCD1 and DCBEXDE2 for error indication
Error Routines
ENDJOB   CLOSE    (ISDATA)
         ...
ISDATA   DCB      DSNAME=INDEXDD,DSORG=IS,MACRF=(GL,SK,PU),                C
               SYNAD=CHECKERR
```

Figure 55. Sequentially Updating an Indexed Sequential Data Set

To update existing records, you must use the READ KU and WRITE K combination. Because READ KU implies that the record will be rewritten in the data set, the system retains the DECB and the buffer used in the READ KU and uses them when the record is written. If you decide not to write the record, you should use the same DECB in another read or write macro instruction or issue a FREEDBUF macro instruction if dynamic buffering was used. If you issue several READ KU or WRITE K macro instructions before checking the first one, you may destroy some of your updated records unless the records are from different blocks.

If there is the possibility that your task and another task will be simultaneously updating the same data set, or the same task has two or more DCBs opened for the same data set, you should use the DCB integrity feature. You specify the DCB integrity feature by coding DISP=SHR in your DD statement. In this way you ensure that the DCB fields are maintained for your program to process the data set correctly. If you do not use DISP=SHR and more than one DCB is open for updating the data set, the results are unpredicatable.

If you specify DISP=SHR, you must also issue an ENQ for the data set before each input/output request and a DEQ upon completion of the request. All users of the data set must use the same *qname* operand for ENQ. For example, the users might use the

data set name as the *qname* operand. For more information about using ENQ and DEQ, see *OS Supervisor Services and Macro Instructions.*

When you are using resume load or the scan mode with QISAM and you want to issue PUTX, issue an ENQ on the data set before processing it and a DEQ after processing is complete. When you are using BISAM to update the data set, do not modify any DCB fields or issue a DEQ until you have issued CHECK or WAIT.

Sharing a BISAM DCB between Related Tasks: When a task using BISAM processes a data set whose DCB is defined and opened by a related task, the task must issue an ENQ on the DCB before an input/output request is issued and must issue a DEQ after the WAIT or CHECK for the input/output request is issued. If the task does not enqueue the DCB and any of its related tasks terminates abnormally, the task may enter a wait state or a program check may occur. See *OS Supervisor Services and Macro Instructions* for more information on the ENQ and DEQ macro instructions and on multitasking.

```
//INDEXDD  DD       DSNAME=SLATE.DICT,DCB=(DSORG=IS,BUFNO=1,...),---
//TAPEDD   DD       ---
ISUPDATE   START    0
           ...
NEXTREC    GET      TPDATA,KEY
           ENQ      (RESOURCE,ELEMENT,E,,SYSTEM)
           READ     DECBRW,KU,,'S',MF=E
           WAIT     ECB=DECBRW
           TM       DECBRW+24,X'FD'              Test for any condition
           BM       RDCHECK                      but overflow
           L        3,DECBRW+16                  Pick up pointer to record
           MVC      30,(20,3),UPDATE             Update record
           WRITE    DECBRW,K,MF=E
           WAIT     ECB=DECBRW
           TM       DECBRW+24,X'FD'              Any errors?
           BM       WRCHECK
           DEQ      (RESOURCE,ELEMENT,SYSTEM)
           B        NEXTREC
RDCHECK    TM       DECBRW+24,X'80'              No record found
           BZ       SYNAD                        If not, go to error routine
           FREEDBUF DECBRW,K,ISDATA              Otherwise, free buffer
           MVC      AREA,KEY
           WRITE    DECBRW,KN,,AREA-16,'S',MF=E  Add record to file
           WAIT     ECB=DECBRW
           TM       DECBRW+24,X'FD'              Test for errors
           BM       SYNAD
           DEQ      (RESOURCE,ELEMENT,,SYSTEM)   Release exclusive control
           B        NEXTREC
           DS       4F                           BISAM WRITE KN work field
AREA       DS       30C                          Logical record to be added
KEY        DS       CL10
UPDATE     DS       CL20
RESOURCE   DC       CL8'SLATE'
ELEMENT    DC       C'DICT'
           READ     DECBRW,KU,ISDATA,'S','S',KEY,MF=L
ISDATA     DCB      DDNAME=INDEXDD,DSORG=IS,MACRF=(RUS,WUA),                C
                 MSHI=INDEX,SMSI=2000
TPDATA     DCB      ---
INDEX      DS       2000C
```

Figure 56. Directly Updating an Indexed Sequential Data Set

Direct Update With Exclusive Control — Indexed Sequential Data Set: In the example shown in Figure 56, the previously described data set is to be updated directly with transaction records on tape. The input tape records are 30 characters long, the key is in positions 1–10, and the update information is in positions 11–30. The update information replaces data in positions 31–50 of the indexed sequential data record.

Exclusive control of the data set is requested since more than one task may be referring to the data set at the same time. Notice that exclusive control is released after each block is written to avoid tying up the data set until the update is completed.

Note the use of the FREEDBUF macro instruction in Figure 56. Usually the FREEDBUF macro instruction has two functions:

- To indicate to the ISAM routines that a record that has been read for update will not be written back
- To free a dynamically obtained buffer

In Figure 56, since the read operation was unsuccessful, the FREEDBUF macro instruction frees only the dynamically obtained buffer.

The first function of FREEDBUF allows you to read a record for update and then decide not to update it without performing a WRITE for update. You can use this function even when your READ macro instruction does not specify dynamic buffering, provided that you have included S (for dynamic buffering) in the MACRF field of your READ DCB.

You can effect an automatic FREEDBUF simply by reusing the DECB, that is, by issuing another READ or a WRITE KN to the same DECB. You should use this feature whenever possible, since it is more efficient than FREEDBUF. For example, in Figure 56, the FREEDBUF macro instruction could be eliminated, since the WRITE KN addressed the same DECB as the READ KU.

For an indexed sequential data set with variable–length records, you may make three types of updates by using the basic access technique. You may read a record and write it back with no change in its length, simply updating some part of the record. You do this with a READ KU followed by a WRITE K, the same way you update fixed–length records. Two other methods for updating variable–length records use the WRITE KN macro instruction and allow you to change the record length.

In one method, a record read for update (by a READ KU) may be updated in a manner that will change the record length and then be written back with its new length by a WRITE KN. In the second method, you may replace a record with another record having the same key and possibly a different length using the WRITE KN macro instruction. To replace a record, it is not necessary to have first read the record.

In either method, when changing the record length, you must place the new length in the DECBLGTH field of the DECB before issuing the WRITE KN macro instruction. If you use a WRITE macro instruction to update a variable–length record that has been marked for deletion, the first bit (no record found) of the exceptional condition code field (DECBEXC1) of the DECB is set on. If this condition is found, the record must be written using a WRITE KN with nothing specified in the DECBLGTH field.

Direct Update — Indexed Sequential Data Set with Variable–Length Records: In Figure 57, an indexed sequential data set with variable–length records is updated directly with transaction records on tape. The transaction records are of variable length and each contains a code identifying the type of transaction. Transaction code 1 indicates that an existing record is to be replaced by one with the same key; 2 indicates that the

record is to be updated by appending additional information, thus changing the record length; 3 or greater indicates that the record is to be updated with no change to its length. For this example, the maximum record length of both data sets is 256 bytes. The key is in positions 6–15 of the records in both data sets. The transaction code is in position 5 of records on the transaction tape. The work area (REPLAREA) size is equal to the maximum record length plus 16 bytes.

```
//INDEXDD  DD       DSNAME=SLATE.DICT,DCB=(DSORG=IS,BUFNO=1,...),---
//TAPEDD   DD       ---
ISUPDVLR  START     0
          ...
NEXTREC   GET       TPDATA,TRANAREA
          CLI       TRANCODE,2                    Determine if replacement or
*                                                 other transaction
          BL        REPLACE                       Branch if replacement
          READ      DECBRW,KU,,'S','S',MF=E       Read record for update
          CHECK     DECBRW,DSORG=IS               Check exceptional conditions
          CLI       TRANCODE,2                    Determine if change or append
          BH        CHANGE                        Branch if change
          ...
          ...
* CODE TO MOVE RECORD INTO REPLACEA+16 AND APPEND DATA FROM TRANSACTION
* RECORD
          ...
          MVC       DECBRW+6(2),REPLAREA+16       Move new length from RDW
*                                                 into DECBLGTH (DECB+6)
          WRITE     DECBRW,KN,,REPLAREA,MF=E      Rewrite record with changed
*                                                 length
          CHECK     DECBRW,DSORG=IS
          B         NEXTREC
CHANGE    ...
          ...
* CODE TO CHANGE FIELDS OR UPDATE FIELDS OF THE RECORD
          ...
          WRITE     DECBRW,K,MF=E                 Rewrite record with no
*                                                 change of length
          CHECK     DECBRW,DSORG=IS
          B         NEXTREC
REPLACE   MVC       DECBRW+6(2),TRANAREA          Move new length from RDW
*                                                 into DECBLGTH (DECB+6)
          WRITE     DECBRW,KN,,TRANAREA+16        Write transaction record
                    ,MF=E                         as replacement for record
*                                                 with the same key
          CHECK     DECBRW,DSORG=IS
          B         NEXTREC
CHECKERR  ...                                     SYNAD routine
          ...
REPLAREA  DS        CL272
TRANAREA  DS        CL4
TRANCODE  DS        CL1
KEY       DS        CL10
TRANDATA  DS        CL241
          READ      DECBRW,KU,ISDATA,'S','S',KEY,MF=L
ISDATA    DECB      DDNAME=INDEXDD,DSORG=IS,MACRF=(RUSC,WUAC),SYNAD=CHECKERR
TPDATA    DCB       ---
```

Figure 57. Directly Updating an Indexed Sequential Data Set with Variable–Length Records

Processing a Direct Data Set

In a direct data set, there is a relationship between a control number or identification of each record and its location on the direct–access volume. This relationship allows you to gain access to a record without an index search. You determine the actual organization of the data set. If the data set has been carefully organized, location of a particular record takes less time than with an indexed sequential data set.

Although you can process a direct data set sequentially using either the queued access technique or the basic access technique, you cannot read record keys using the queued access technique. When you use the basic access technique, each unit of data transmitted between main storage and an I/O device is regarded by the system as a record. If, in fact, it is a block, you must perform any blocking or deblocking required. For that reason, the BLKSIZE value must be equal to the LRECL value when format–F or format–U records are processed. When format–V records are used, the BLKSIZE value must be equal to the LRECL value plus 4. Only BLKSIZE must be specified when you add or update records on a direct data set.

As indicated in the discussion of direct–access devices, record keys are optional. If they are specified, they must be used for every record and must be of a fixed length.

Organizing a Direct Data Set

In developing the organization of your data set, you can use *direct addressing*. When direct addresses are used, the location of each record in the data set is known.

If format–F records with keys are being written, the key of each record can be used to identify the record. For example, a data set with keys ranging from 0 to 4999 should be allocated space for 5000 records. Each key relates directly to a location that you can refer to as a relative record number. The main disadvantage of this type of organization is that records may not exist for many of the keys even though space has been reserved for them.

Space could be allocated on the basis of the number of records in the data set rather than on the range of keys. This type of organization requires the use of a cross–reference table. When a record is written in the data set, you must note the physical location either as an actual address or as a relative track and record number. The addresses must then be stored in a table that is searched when a record is to be retrieved. Disadvantages are that cross–referencing can be used efficiently only with a small data set, storage is required for the table, and processing time is required for searching and updating the table.

A more common, but somewhat complex, technique for organizing the data set involves the use of indirect addressing. In indirect addressing, the address of each record in the data set is determined by a mathematical manipulation of the key. This manipulation is referred to as randomizing or conversion. Since a number of randomizing procedures could be used, no attempt is made here to describe or explain those that might be most appropriate for your data set.

Referring to a Record in a Direct Data Set

Once you have determined how your data set is to be organized, you must consider how the individual records will be referred to when the data set is updated or new records are added. This is important for determining whether a return address will be

required when the data is created and, if so, in what form the return address will be used. The record identification can be represented in any of the following forms:

Relative Block Address: You specify the relative location of the record (block) within the data set as a 3–byte binary number. This type of reference can be used only with format–F records. The system computes the actual track and record number. The relative block address of the first block is 0.

Relative Track Address: You specify the relative track as a 2–byte binary number and the actual record number on that track as a 1–byte binary number. The relative track address of the first track is 0.

Relative Track Address and Actual Key: In addition to the relative track address, you specify the address of a main–storage location containing the record key. The system computes the actual track address and searches for the record with the correct key.

Actual Address: You supply the actual address in the standard 8–byte form — MBBCCHHR. Remember that the use of an actual address may force you to indicate that the data set is unmovable.

Extended Search: You request that the system begin its search with a specified starting location and continue for a certain number of records or tracks. This same option can be used to request a search for unused space in which a record can be added.

To use the extended search option, you must indicate in the DCB the number of tracks (including the starting track) or records (including the starting record) that are to be searched. If you indicate a number of records, the system may actually examine more than this number. In searching a track, the system searches the whole track (starting with the first record); it therefore may examine records that precede the starting record or follow the ending record.

If the DCB specifies a number equal to or greater than the number of tracks allocated to the data set or the number of records within the data set, the entire data set is searched in the attempt to satisfy your request.

Exclusive Control for Updating: If more than one task in the same job step is referring to the same data set through the same DCB, exclusive control can be requested in the DCB macro instruction to prevent simultaneous reference to the same record. No other task in the system requesting exclusive control of that record is given access to it until it is released by means of a WRITE or RELEX macro instruction. Only the task that opened the data set can use the RELEX macro instruction to release exclusive control of a record within that data set.

Creating a Direct Data Set

Once the organization of a direct data set has been determined, the process of creating it is almost identical to that of creating a sequential data set. The data set organization field in the DCB macro instruction is specified as physical sequential (DSORG=PS or PSU). The DD statement must indicate direct–access (DSORG=DA or DAU). The DCB macro instruction must specify a direct–access device (DEVD=DA). If keys are used, a key length (KEYLEN) must also be specified. Record length (LRECL) should not be specified. The WRITE macro instruction should be of the form used to create a direct data set (MACRF=WL).

If you are using direct addressing with keys, you can reserve space for future format–F records by writing a dummy record. To reserve or truncate a track f or format–U or

format–V records, write a capacity record (see "Direct–Access Device Characteristics").

Format–F records are written sequentially as they are presented. When a track is filled, the system automatically writes the capacity record and advances to the next track. Because of the form in which relative track addresses are recorded, direct data sets whose records are to be identified by means other than actual address must be limited in size to no more than 65,536 tracks for the entire data set.

Tape–to–Disk — Direct Data Set: In the example problem in Figure 58, a tape containing 204–character records arranged in key sequence is used to create a direct data set. A 4–byte binary key for each record ranges from 1000 to 8999, so space for 8000 records is requested.

```
//DAOUTPUT DD       DSNAME=SLATE.INDEX.WORDS,DCB=(DSORG=DA,                  C
               BLKSIZE=200,KEYLEN=4,RECFM=F),SPACE=(204,8000),---
//TAPINPUT DD       ---
DIRECT     START
           ...
           L        9,=F'1000'
           OPEN     (DALOAD,(OUTPUT),TAPEDCB)
           LA       10,COMPARE
NEXTREC    GET      TAPEDCB
           LR       2,1
COMPARE    C        9,0(2)                    Compare key of input against
*                                             control number
           BNE      DUMMY
           WRITE    DECB1,SF,DALOAD,(2)       Write data record
           CHECK    DECB1
           AH       9,=H'1'
           B        NEXTREC
DUMMY      C        9,=F'8999'                Have 8000 records been written?
           BH       ENDJOB
           WRITE    DECB2,SD,DALOAD,DUMAREA   Write dummy
           CHECK    DECB2
           AH       9,=H'1'
           BR       10
INPUTEND   LA       10,DUMMY
           BR       10
ENDJOB     CLOSE    (TAPEDCB,,DALOAD)
           ...
DUMAREA    DS       CL5
DALOAD     DCB      DSORG=PS,MACRF=(WL),DDNAME=DAOUTPUT,                     C
                    DEVD=DA,SYNAD=CHECKER,---
TAPEDCB    DCB      EODAD=INPUTEND,MACRF=(GL), ---
```

Figure 58. Creating a Direct Data Set

Adding or Updating Records on a Direct Data Set

The techniques for adding records to a direct data set depend on the format of the records and the organization used.

Format–F With Keys: Adding a record amounts to essentially an update by record identification. The reference to the record can be made by either a relative block address or a relative track address.

If you attempt to add a record by relative block address, the system converts the address to a relative track address. That track is searched and the new record written in place of the first dummy record on the track. If there is no dummy record on the track, you are informed that the write operation did not take place. If you request the extended search option, the new record will be written in place of the first dummy record found within the search limits you specify. If none is found, you are notified that the write operation could not take place. In the same way, a reference by relative track address causes the record to be written in place of the first dummy record on that track or the first within the search limits, if requested.

Format–F Without Keys: Here too, adding a record is really updating a dummy record already in the data set. The main difference is that dummy records cannot be written automatically when the data set is created. You will have to use your own method for flagging dummy records. The update form of the WRITE macro instruction (MACRF=W) must be used rather than the add form (MACRF=WA).

You will have to retrieve the record first (using a READ macro instruction), test for a dummy record, update, and write.

Format–V or Format–U With Keys: The technique used to add records in this case depends on whether records are located by indirect addressing or a cross–reference table. If indirect addressing is used, you must at least initialize each track (write a capacity record) even if no data is actually written. That way the capacity record indicates how much space is available on the track. If a cross–reference table is used, you should exhaust the input and then initialize enough succeeding tracks to contain any additions that might be required.

To add a new record, use a relative track address. The system examines the capacity record to see if there is room on the track. If there is, the new record is written. Under the extended search option, the record is written in the first available area within the search limit.

Format–V or Format–U Without Keys: Because a record of this type does not have a key, you can refer to the record only by its relative track or actual address. When you add a record to this data set, you must update the cross–reference table required by a data set of this type.

Tape–to–Disk Add — Direct Data Set: The example in Figure 59 involves adding records to the data set created in the last example. Notice that the write operation adds the key and the data record to the data set. If the existing record is not a dummy record, an indication is returned in the exception code of the DECB. For that reason, it is better to use the WAIT macro instruction instead of the CHECK macro instruction to test for errors or exceptional conditions.

Tape–to–Disk Update — Direct Data Set: The example in Figure 60 is similar to that in Figure 59, but involves updating rather than adding. There is no check for dummy records. The existing direct data set contains 25,000 records whose 5–byte keys range from 00001 to 25000. Each data record is 100 bytes long. The first 30 characters are to be updated. Each input tape record consists of a 5–byte key and a 30–byte data area. Notice that only data is brought into main storage for updating.

```
//DIRADD   DD       DSNAME=SLATE.INDEX.WORDS,---
//TAPEDD   DD       ---
DIRECTAD  START
          ...
          OPEN     (DIRECT,(OUTPUT),TAPEIN)
NEXTREC   GET      TAPEIN,KEY
          L        4,KEY                          Set up relative record number
          SH       4,=H'1000'
          ST       4,REF
          WRITE    DECB,DA,DIRECT,DATA,'S',KEY,REF+1
          WAIT     ECB=DECB
          CLC      DECB+1(2),=X'0000'             Check for any errors
          BE       NEXTREC
```

Check error bits and take required action

```
DIRECT    DCB      DDNAME=DIRADD,DSORG=DA,RECFM=F,KEYLEN=4,BLKSIZE=200,       C
                MACRF=(WA)
TAPEIN    DCB      ---
KEY       DS       F
DATA      DS       CL200
REF       DS       F
          ...
```

Figure 59. Adding Records to a Direct Data Set

```
//DIRECTDD DD       DSNAME=SLATE.INDEX.WORDS,---
//TAPINPUT DD       ---
DIRUPDAT  START
          ...
          OPEN     (DIRECT,(UPDAT),TAPEDCB)
NEXTREC   GET      TAPEDCB,KEY
          PACK     KEY,KEY
          CVB      3,KEYFIELD
          SH       3,=H'1'
          ST       3,REF
          READ     DECBRD,DI,DIRECT,'S','S',0,REF+1
          CHECK    DECBRD
          L        3,DECBRD+12
          MVC      0(30,3),DATA
          ST       3,DECBWR+12
          WRITE    DECBWR,DI,DIRECT,'S','S',0,REF+1
          CHECK    DECBWR
          B        NEXTREC
          ...
KEYFIELD  DS       0D
          DC       XL3'-'
KEY       DS       CL5
DATA      DS       CL30
REF       DS       F
DIRECT    DCB      DSORG=DA,DDNAME=DIRECTDD,MACRF=(RISC,WIC),    C
               OPTCD=R,BUFNO=1,BUFL=100
TAPEDCB   DCB      ---
          ...
```

Figure 60. Updating a Direct Data Set

Consideration for User Labels: User labels must be created when the data set is created. They may be updated, but not added or deleted, during processing of a direct data set. When creating a multivolume direct data set using BSAM, you should turn off the header exit entry after OPEN and turn on the trailer label exit entry just before issuing the CLOSE. This eliminates the end–of–volume exits. The first volume, containing the user label track, must be mounted when the data set is closed. If you have requested exclusive control, OPEN and CLOSE will ENQ and DEQ to prevent simultaneous reference to user labels.

PART 3: DATA SET DISPOSITION AND SPACE ALLOCATION

Allocating Space on Direct–Access Volumes

When direct–access storage space is required for a data set, you specify the amount of space needed and the device type, and the operating system selects the device and allocates the space accordingly. This arrangement provides for flexible and efficient use of devices and available storage space, and relieves you of considering the details involved in efficient space control.

Before a direct–access volume can be used for data storage, it must be initialized by the utility program IBCDASDI. The IBCDASDI functions include in part:

- Creating the standard 80–byte volume label and writing it on cylinder 0, track 0, of the volume.
- Initializing the volume table of contents (VTOC). The location of the VTOC depends on the conventions your installation uses in initializing the volume.
- Writing the home address (HA) and capacity record (R0) for each track.
- Checking tracks and making alternate track assignments if necessary.

When the data set is to be stored on a direct–access volume, you must supply, in the DD statement, control information designating the amount of space to be allocated and the manner in which it is to be allocated.

Specifying Space Requirements

The amount of space required can be specified in blocks, tracks, or cylinders. If you want to maintain device independence, specify your space requirements in blocks. If your request is in tracks or cylinders, you must be aware of such device considerations as cylinder and track capacity.

Cylinder allocation allows faster input/output of sequential data sets than does track allocation. Track allocation stops input/output at the end of every track to prevent references on the same cylinder outside of the data set. The time difference occurs when you use the sequential access method or the partitioned access method to read a data set whose record format is not fixed standard (FS). If the data set is partitioned, the time difference occurs during both loading of a module from the data set and reading of the data set's directory.

Allocation by Blocks: When the amount of space required is expressed in blocks, you must specify the number and average length of the blocks within the data set, as in this example:

```
//       DD   --,SPACE=(300,(5000,100))
```

300 = average block length in bytes
5000 = quantity (number of blocks)
100 = increment (to be used if the quantity is not sufficient) allocated in additional blocks

Note that when average block length and secondary space allocation are being used, the BLKSIZE parameter specified must be equal to the maximum block length.

From this information, the operating system estimates and allocates the number of tracks required. Space is always in whole tracks. You may also request that the space allocated for a specific number of blocks begin and end on cylinder boundaries.

You must be certain that both the quantity and the increment are large enough to contain the largest block to be written. Otherwise, all of the space requested is allocated but erased as the system tries to find a space large enough for the record.

Allocation by Tracks or Cylinders: When the amount of space required is expressed in tracks or cylinders, you must also specify the device type in the DD statement, as in these examples:

```
//        DD   --,SPACE=(TRK,(100,5)),UNIT=2301
//        DD   -,SPACE=(CYL,(3,1)),UNIT=2311
```

Allocation by Absolute Address: If the data set contains location–dependent information in the form of an absolute track address (MBBCCHHR), space should be requested with respect to the number of tracks and the beginning address, as in this example:

```
//        DD   -,SPACE=(ABSTR,(500,20)),UNIT=2311
```

where 500 tracks are required, beginning at relative track 20.

Additional Space Allocation Options: The DD statement provides you with a great deal of flexibility in specifying space requirements. You can request that the space be continuous (SPACE=CONTIG) or not (SPACE=ALX). These and other options are described in detail in *OS Job Control Language Reference.*

Estimating Space Requirements

To determine how much space your data set requires, you must consider these variables:

- Device type
- Track capacity
- Tracks per cylinder
- Cylinders per volume
- Data length (blocksize)
- Key length
- Device overhead

Figure 61 lists the physical characteristics of a number of direct–access storage devices.

The term *device overhead* refers to the space required on each track for hardware data, that is, address markers, count areas, gaps between records, record 0, etc. Device overhead varies with each device and depends also on whether the blocks are written with keys. To compute the actual space required for each block including device overhead, you can use the formulas in Figure 62. Note that any fraction of a byte must be treated as an extra byte. For example, if the formulas give 15.067 bytes, you must allocate 16 bytes.

Device Type	Volume Type	Track Capacity[1]	Tracks Per Cylinder	Number of Cylinders	Total Capacity[1]
2311	Disk	3625	10	200	7,250,000
2314[2]	Disk	7294	20	200	29,176,000
2302	Disk	4984	46	246	56,398,944
3330	Disk	13030	19	404	101,751,270
2303	Drum	4892	10	80	3,913,600
2305–1	Drum	14136	8	48	5,428,224
2305–2	Drum	14660	8	96	11,258,880
2301	Drum	20483	8	25[3]	4,096,600
2321	Cell	2000	20[4]	980[4]	39,200,000

[1] Capacity is indicated in bytes.
[2] Data applies also to the 2319 Disk Storage Device.
[3] There are 25 logical cylinders in a 2301 Drum.
[4] A volume is equal to the one bin in a 2321 Data Cell.

Figure 61. Direct–Access Storage Device Capacities

	Bytes Required by Each Data Block			
Device	Blocks With Keys		Blocks Without Keys	
	Bi	Bn	Bi	Bn
2311	81+(KL+DL)537/512	20+KL+DL	61+(DL)537/512	DL
2314/2319	146+(KL+DL)534/512	45+KL+DL	101+(DL)534/512	DL
2302	81+(KL+DL)537/512	20+KL+DL	61+(DL)537/512	DL
3330	191+KL+DL	191+KL+DL	135+DL	135+DL
2303	146+KL+DL	38+KL+DL	108+DL	DL
2301	186+KL+DL	53+KL+DL	133+DL	DL
2305–1	632+KL+DL	632+KL+DL	430+DL	430+DL
2305–2	289+KL+DL	289+KL+DL	198+DL	198+DL
2321	100+(KL+DL)537/512	16+KL+DL	84+(DL)537/512	DL

Bi is any block but the last on the track.
Bn is the last block on the track.
DL is data length.
KL is key length.

Figure 62. Direct–Access Device Overhead Formulas

The formulas can be combined in the following way:

If you intend to specify your space requirements in tracks (TRK) or cylinders (CYL), your estimate should be made as shown above. If you request absolute tracks (ABSTR), remember that you cannot allocate track 0, cylinder 0. The amount of space required for the VTOC will reduce the space available on the rest of the volume.

If you specify your space requirements in average block length, the system performs the computations for you.

Because a sequential data set and a direct data set are created in the same way, the estimate and specification of space requirements are identical. If you use the WRITE SZ macro instruction, your secondary allocation for a direct data set should be at least 2 tracks. Space allocation for a partitioned data set requires that you also consider the space used for the directory. Similarly, allocation for an indexed sequential data set requires that you consider the space needed for the prime area, index areas, and overflow areas.

Allocating Space for a Partitioned Data Set

What is the average size of the members to be stored on your direct–access volume? How many members will fit on the volume? Will you need directory entries for the member names only or will aliases be used? How many? Will members be added or replaced frequently? All of these questions must be answered if you are to estimate your space requirements accurately and use the space efficiently. Note, too, that a partitioned data set cannot extend beyond one volume.

If your data set will be quite large, or you expect to do a lot of updating, it might be best to allocate a full volume. If it will be small or seldom subject to change, you should make your estimate as accurate as possible to avoid wasted space or wasted time used for recreating the data set.

Because the characteristics of all the members of the data set must be uniform, the record format could be specified as undefined (RECFM=U) and the blocksize (BLKSIZE) as a maximum length. It is a good practice to indicate a block length equal to track capacity, for example, BLKSIZE=3625 for a 2311 disk. You might then ask for either 200 tracks, or 20 cylinders, thus allowing for 725,000 bytes of data.

Assuming an average length of 70,000 bytes for each member, you need space for at least 10 directory entries. If each member also has an average of three aliases, space for an additional 30 directory entries is required.

Space for the directory is expressed in 256–byte blocks. Each block contains from 3 to 20 entries, depending on the length of the user data field. If you expect 40 directory entries, request at least 8 blocks. Because the space for the directory is allocated in full tracks, any unused space on the track is wasted unless there is enough space left to contain a block of the first member. Therefore, the most advisable request in this case would be for 10 blocks.

Any of the following space specifications would cause the same allocation:

SPACE=(3625,(200,,10))

SPACE=(CYL,(20,,10))

SPACE=(TRK,(200,,10))

Although an increment has been omitted in these examples, it could have been supplied to provide for extension of the member area. The directory size, however, cannot be extended.

Allocating Space for an Indexed Sequential Data Set

An indexed sequential data set has three areas: prime, index, and overflow. Space for these areas can be subdivided and allocated as follows:

- Prime area — If you request a prime area only, the system automatically uses a portion of that space for indexes, taking one cylinder at a time as needed. Any unused space in the last cylinder used for index will be allocated as an independent overflow area. More than one volume can be used in most cases, but all volumes must be for devices of the same device type.
- Index area — You can request that a separate area be allocated to contain your cylinder and master indexes. The index area must be contained within one volume, but this volume can be on a device of a different type than the one that

contains the prime area volume. If a separate index area is requested, you cannot catalog the data set with a DD statement.

If the total space occupied by the prime area and index area does not exceed one volume, you can request that the separate index area be embedded in the prime area (to reduce access arm movement) by indicating an index size in the SPACE parameter of the DD statement defining the prime area.

If you request space for prime and index areas only, the system automatically uses any space remaining on the last cylinder used for master and cylinder indexes for overflow, provided the index area is on a device of the same type as the prime area.

- Overflow area — Although you can request an independent overflow area, it must be contained within one volume. If no specific request for index area is made, then it will be allocated from the specified independent overflow area.

  To request that a designated number of tracks on each cylinder be used for cylinder overflow records, you must use the CYLOFL parameter of the DCB macro instruction. The number of tracks that you can use on each cylinder equals the total number of tracks on the cylinder minus the number of tracks needed for track index and for prime data, that is:

  Usable tracks = total tracks – (track index tracks + prime data tracks)

Note that when you create a 1–cylinder data set, ISAM reserves 1 track on the last cylinder for the end–of–file filemark.

When you request space for an indexed sequential data set, the DD statement must follow a number of conventions, as shown below and summarized in Figure 63.

- Space can be requested only in cylinders (CYL) or absolute tracks (ABSTR). If the absolute track technique is used, the designated tracks must make up a whole number of cylinders.
- Data set organization (DSORG) must be specified as indexed sequential (IS or ISU) in both the DCB macro instruction and the DCB parameter of the DD statement.
- All required volumes must be mounted when the data set is opened; that is, volume mounting cannot be deferred.
- If your prime area extends beyond one volume, you must indicate the number of units and volumes to be spanned, for example, UNIT=(2311,3),VOLUME=(,,,3).
- You can catalog the data set using the DD statement parameter DISP=(,CATLG) only if the entire data set is defined by one DD statement, that is, if you did not request a separate index or independent overflow area.

As your data set is created, the operating system builds the track indexes in the prime data area. Unless you request a separate index area or an embedded index area, the cylinder and master indexes are built in the independent overflow area. If you did not request an independent overflow area, the cylinder and master indexes are built in the prime area.

Criteria			Restrictions on Unit Types and Number of Units Requested	Resulting Arrangement of Areas
1. Number of DD Statements	2. Types of DD Statements	3. Index Size Coded?		
3	INDEX PRIME OVFLOW	–	None	Separate index, prime, and overflow areas.
2	INDEX PRIME	–	None	Separate index and prime areas.
2	PRIME OVFLOW	No	None	Prime area and overflow area with an index at its end
2	PRIME OVFLOW	Yes	The statement defining the prime area cannot request more than one unit.	Prime area and embedded index, and overflow area.
1	PRIME	No	None	Prime area with index at its end. Any unused index area is used for indepen-dent overflow.
1	PRIME	Yes	Statement cannot request more than one unit.	Prime area with embedded index area.

Figure 63. Requests for Indexed Sequential Data Sets

If an error is encountered during allocation of a multivolume data set, the IEHPROGM utility program should be used to scratch the DSCBs of the data sets that were successfully allocated. The IEHLIST utility program can be used to determine whether or not part of the data set has been allocated. The IEHLIST utility program is also useful to determine whether space is available or whether identically named data sets exist before space allocation is attempted for indexed sequential data sets. These utility programs are described in *OS Utilities*.

Specifying a Prime Data Area

To request that the system allocate space and subdivide it as required, you should code:

```
//ddname    DD  DSNAME=dsname,DCB=DSORG=IS,                    C
//              SPACE=(CYL,quantity,,CONTIG),UNIT=unitname,    C
//              DISP=(,KEEP),---
```

You can accomplish the same type of allocation by qualifying your *dsname* with the element indication (PRIME). This element is assumed if omitted. It is required only if you request an independent index or overflow area. To request an embedded index area when an independent overflow area is specified, you must indicate DSNAME=dsname(PRIME). To indicate the size of the embedded index, you specify SPACE=(CYL,(quantity,,index size)).

Specifying a Separate Index Area

To request a separate index area, other than an embedded area as described above, you must use a separate DD statement. The element name is specified as (INDEX). The

space and unit designations are as required. Notice that only the first DD statement can have a data definition name. The data set name *(dsname)* must be the same.

```
//ddname     DD   DSNAME=dsname(INDEX),---
//            DD   DSNAME=dsname(PRIME),---
```

Specifying an Independent Overflow Area

A request for an independent overflow area is essentially the same as for a separate index area. Only the element name, OVFLOW, is changed. If you do not request a separate index area, only two DD statements are required.

```
//ddname     DD   DSNAME=dsname(INDEX,---
//            DD   DSNAME=dsname(PRIME),---
//            DD   DSNAME=dsname(OVFLOW),---
```

Calculating Space Requirements for an Indexed Sequential Data Set

To determine the number of cylinders required for an indexed sequential data set, you must consider the number of blocks that will fit on a cylinder, the number of blocks that will be processed, and the amount of space required for indexes and overflow areas. In making the computations, consider additional space that is required for device overhead as shown in Figure 62. Remember the formula:

Blocks per track = 1 +((Track capacity − Length of the last block)/(Length of other blocks))

$$Bt = 1 + ((Ct - Bn)/Bi)$$

The following eight steps summarize calculation of space requirements for an indexed sequential data set.

Step 1

Once you know how many records will fit on a track and the maximum number of records you expect to create, you can determine how many tracks you will need for your data.

Number of tracks required = (Maximum number of blocks/Blocks per track) + 1

ISAM load mode reserves the last prime data track for the filemark.

Example: Assume that a 200,000 record part–of–speech dictionary is stored on an IBM 2311 Disk Storage Drive as an indexed sequential data set. Each record in the dictionary has a 12–byte key (the word itself) and an 8–byte data area containing a part–of–speech code and control information. Each block contains 50 records; LRECL=20 and BLKSIZE=1000. Using the formula from Figure 62, we find that each track will contain 3 blocks or 150 records. A total of 1333 1/3 tracks will be required for the dictionary.

$$Bt = 1 + \frac{3625 - (20 + 12 + 1000)}{81 + 1.049(12 + 1000)} = 1 + \frac{2593}{1143} = 3$$

Records per track = (3 blocks)(50 records per block) = 150

$$\text{Prime data tracks required (T)} = \frac{200{,}000 \text{ records}}{150 \text{ records per track}} + 1 = 1334\ 1/3$$

Step 2

You will want to anticipate the number of tracks required for cylinder overflow areas. The computation is the same as for prime data tracks, but you must remember that overflow records are unblocked and a 10–byte link field is added. Remember, if you exceed the space allocated for any cylinder overflow area, an independent overflow area is required. Those records are not placed in another cylinder overflow area.

$$\text{Overflow records per track (Ot)} = 1 + \frac{\text{Track capacity} - \text{Length of last overflow record}}{\text{Length of other overflow records}}$$

$$Ot = 1 + ((Ct - Rn)/Ri)$$

Example: Approximately 5000 overflow records are expected for the data set described in step 1. Since 29 overflow records will fit on a track, 173 overflow tracks are required. This is approximately 2 overflow tracks for every 15 prime data tracks. Since the 2311 disk has 10 tracks per cylinder, it would probably be best to allocate 2 tracks per cylinder for overflow.

$$Ot = 1 + \frac{3625 - (20 + 12 + 20 + 10)}{81 + 1.049(12 + 20 + 10)} = 1 + 3563 = \frac{29}{126}$$

$$\text{Overflow tracks required} = \frac{\text{5000 records}}{\text{29 records per track}} = 173$$

Overflow tracks per cylinder (Oc) = 2

Step 3

You will have to set aside space in the prime area for track index entries. There will be two entries (normal and overflow) for each track on a cylinder that contains prime data records. The data field of each index entry is always 10 bytes long. The key length corresponds to the key length for the prime data records. How many index entries will fit on a track?

$$\text{Index entries per track (It)} = 1 + \frac{\text{Track capacity} - \text{Length of last index entry}}{\text{Length of other index entries}}$$

$$It = 1 + ((Ct - En)/Ei)$$

Example: Again assuming 2311 disk and records with 12–byte keys, we find that 35 index entries will fit on a track.

$$It = 1 + \frac{3625 - (20 + 12 + 10)}{81 + 1.049(12 + 10)} = 1 + \frac{3583}{105} = 1 + 34 = 35$$

Step 4

The number of tracks required for track index entries will depend on the number of tracks per cylinder and the number of track index entries per track. Any unused space on the last track of the track index can be used for any prime data records that will fit.

$$\text{Number of track index tracks per cylinder (Ic)} = \frac{2(\text{Tracks per cylinder} - \text{overflow tracks per cylinder}) + 1}{\text{Index entries per track} + 2}$$

$$Ic = (2(Tc - Oc) + 1)/(It + 2)$$

Note that for variable–length records or when a prime data record will not fit on the last track of the track index, the last track of the track index is not shared with prime data records. In such a case, if the remainder of the division is less than or equal to 2, do not round the quotient up to the nearest integer. In all other cases, round the quotient up to the nearest integer.

Example: The 2311 disk has 10 tracks per cylinder. You can fit 35 track index entries per track. Therefore, you need less than 1 track for each cylinder:

$$Ic = \frac{2(10-2)+1}{35+2} = \frac{17}{37}$$

The space remaining on the track is ((1–17/37)(3625)) = 1960 bytes. This is enough for 1 block of prime data records. Since the normal number of blocks per track is 3, the block uses 1/3 of the track, and the effective value of Ic is therefore 1–1/3 = 2/3.

Step 5

Next you have to compute the number of tracks available on each cylinder for prime data records. You cannot include tracks set aside for cylinder overflow records.

$$\text{Prime data tracks per cylinder} = \left(\begin{matrix}\text{Tracks}\\ \text{per cylinder}\end{matrix}\right) - \left(\begin{matrix}\text{Overflow tracks}\\ \text{per cylinder}\end{matrix}\right) - \left(\begin{matrix}\text{Index tracks}\\ \text{per cylinder}\end{matrix}\right)$$

$$Pc = Tc - Oc - Ic$$

Example: If you set aside 2 cylinder overflow tracks, and you require 2/3 of a track for the track index, 7 1/3 tracks are available on each cylinder for prime data records.

$$Pc = 10 - 2 - 2/3 = 7\ 1/3$$

Step 6

The number of cylinders required for the prime data records, track index area, and cylinder overflow area is determined by the number of prime data tracks required divided by the number of prime data tracks available on each cylinder.

$$\text{Number of cylinders required} = \text{Prime data tracks required/Prime data tracks per cylinder}$$

$$C = T/Pc$$

Example: You need 1333 1/3 tracks for prime data records. You can use 7 1/3 tracks per cylinder. Therefore, 182 cylinders are required for your prime area and cylinder overflow areas.

$$C = (1333\ 1/3)/(7\ 1/3) = 181.9$$

Step 7

You will need space for a cylinder index as well as track indexes. There is a cylinder index entry for each track index (for each cylinder allocated for the data set). The size of each entry is the same as the size of the track index entries; therefore, the number of entries that will fit on a track is the same as the number of track index entries. Unused space on a cylinder index track is not shared.

Number of tracks
required for = (Track indexes + 1)/Index entries per track
cylinder index

Ci = (C+1)/It

Example: You have 182 track indexes. Since 35 index entries fit on a track, you need 5.3 tracks for your cylinder index. The remaining space on the last track is unused.

Ci = (182 + 1)/35 = 5.3

Note that every time a cylinder index crosses a cylinder boundary, ISAM writes a dummy index entry that lets ISAM chain the index levels together. The addition of dummy entries can increase the number of tracks required for a given index level. To determine how many dummy entries will be required, divide the total number of tracks required by the number of tracks on a cylinder. If the remainder is 0, subtract 1 from the quotient. If the corrected quotient is not 0, calculate the number of tracks these dummy entries require. Also consider any additional cylinder boundaries crossed by the addition of these tracks and by any track indexes starting and stopping within a cylinder.

Step 8

If you have a data set large enough to require master indexes, you will want to calculate the space required according to the number of tracks for master indexes (NTM parameter) you specified in the DCB macro instruction or the DD statement.

If the cylinder index exceeds the NTM specification, an entry is made in the master index for each track of the cylinder index. If the master index itself exceeds the NTM specification, a second–level master index is started. Up to three levels of master indexes are created if required.

The space requirements for the master index are computed in the same way as those for the cylinder index.

Number of tracks
required for = (Number of cylinder index tracks + 1)/Index entries per track
master indexes

M_1 = (Ci+1)/It when Ci>NTM

M_2 = (M_1+1)/It when M_1>NTM

M_3 = (M_2+1)/It when M_2>NTM

Example: Assume that your cylinder index will require 22 tracks. Since large keys are used, only 10 entries will fit on a track. Assuming that NTM was specified as 2, 3 tracks will be required for a master index, and two levels of master index will be created.

M_1 = (22+1)/10 = 2.3

Note that every time a master index crosses a cylinder boundary, ISAM writes a dummy index entry that lets ISAM chain the index levels together. The addition of dummy entries can increase the number of tracks required for a given index level. To determine how many dummy entries will be required, divide the total number of tracks required by the number of tracks on a cylinder. If the remainder is 0, subtract 1 from the quotient. If the corrected quotient is not 0, calculate the number of tracks these dummy entries require. Also consider any additional cylinder boundaries crossed by

the addition of these tracks and by any track indexes starting and stopping within a cylinder.

Summary: Indexed Sequential Space Requirement Calculations

1. How many blocks will fit on a track?

   $Bt = 1 + ((Ct-Bn)/Bi)$

2. How many overflow records will fit on a track?

   $Ot = 1 + ((Ct-Rn)/Ri)$

3. How many index entries will fit on a track?

   $It = 1 + ((Ct-En)/Ei)$

4. How many track index tracks are needed per cylinder?

   $Ic = (2(Tc-Oc)+1)/(It+2)$

5. How many tracks on each cylinder can be used for prime data records?

   $Pc = Tc - Oc - Ic$

6. How many cylinders are needed for the prime data area?

   $C = T/Pc$

7. How many tracks are required for the cylinder index?

   $Ci = (C+1)/It$

8. How many tracks are required for master indexes?

   $M = (Ci+1)/It$

Control and Disposition of Data Sets

You specify two kinds of status and disposition information for the data sets you use for your processing by coding DISP=(status,disposition) in the disposition field of the DD statement. The first kind deals with the status of the data set when you begin processing and the relationship of the data set to other job steps in your job or other jobs. The second deals with what is to be done with the data set when you have completed processing. In the latter case, you can take advantage of the catalog of the operating system.

A data set that is being used for input has a status of OLD. If it can be used by more than one job, the status should be specified as SHR. If you are going to add to the input data set, specify MOD. The system automatically positions the access mechanism after the last record when the data set is opened. A new output data set should be indicated as NEW.

Having identified the status of the data set at the beginning of your job step, you should specify how you want it disposed of at the end of processing. If the disposition is to be unchanged, you need not specify anything. The status of an existing data set remains unchanged; a new data set is deleted.

The requested disposition is performed at the end of the job step. A data set to be used in a later job can be kept (KEEP) until a subsequent request is made to delete it. If the data set is to be used by more than one job step in the same job, you can specify that it is to be passed (PASS).

If you specify the CATLG disposition, the data set name is recorded in the catalog by the system and its volume is noted. An old data set can subsequently be removed from the catalog if you specify UNCATLG.

If you wish, you can specify one disposition to be performed if the job step terminates normally, and a different disposition to be performed if the job step terminates abnormally. For example, you can specify DISP=(OLD,DELETE,KEEP) if you wish to delete a data set under normal conditions, but wish to keep it if processing is abnormally terminated. For normal termination, you can specify any disposition — PASS, KEEP, DELETE, CATLG, or UNCATLG; for abnormal termination, you can specify any disposition except PASS.

Routing Data Sets through the Output Stream

Whenever you have an output data set to be printed or punched, you can route the data set through the output stream. Data sets in the output stream are written into intermediate storage on a direct–access device and later transferred to the card punch or printer by a system routine called the system output writer. Routing data sets through the output stream improves operating–system efficiency because the unit–record device is not tied up for the entire length of your program. It is busy only as long as it takes the system output writer to punch or print your output.

When you route a data set through the output stream, you do not request a unit–record device for exclusive use by your job step. Instead, you request an output class that is assigned to the device you need. You should have a list of the output classes in your installation and of the devices assigned to each class.

Output classes are identified by the letters A–Z and the digits 0–9. You request an output class by coding the SYSOUT keyword parameter in your DD statement. For example, code SYSOUT=A to request output class A. You can assign several data sets to the same output class. The system output writer copies data sets in the order of their DD statements. For other parameters you can code with the SYSOUT parameter, see "The SYSOUT Parameter" in the section "The DD Statement" in *OS Job Control Language Reference.*

You open and close a SYSOUT data set (a data set routed through the output system) in the same way as any other data set. If specified in an exit list, the DCB exit routine is entered in the usual manner.

You create a SYSOUT data set by using either the basic sequential access method (BSAM) or the queued sequential access method (QSAM). You can write records in any format defined for the type of unit–record device to which the data set will be transferred. Record length must not exceed the maximum allowable for the device.

When you use QSAM with fixed–length blocked records or BSAM, the DCB blocksize parameter does not have to be a multiple of logical record length (LRECL) if the blocksize is specified through the SYSOUT DD statement. Under these conditions, if blocksize is greater than LRECL but not a multiple of LRECL, blocksize is reduced to the nearest lower multiple of LRECL when the data set is opened. This feature allows a cataloged procedure to specify blocking for SYSOUT data sets, even though your LRECL is not known to the system until execution. Therefore, the SYSOUT DD

statement of the go step of a compile–load–go procedure can specify blocksize without blocksize being a multiple of LRECL. For further information, refer to "The DCB Parameter" in the section "The DD Statement" in *OS Job Control Language Reference*.

Because a SYSOUT data set is written on a direct–access device, you should omit the DEVD operand in the DCB macro instruction, or should code DEVD=DA.

Your SYNAD routine is entered on errors that occur when you write the data set into intermediate storage on a direct–access device.

Your program is responsible for printing format, pagination, and header control. Use of control characters must be indicated in the usual way in the DCB. If you do not use control characters, a standard control is supplied. When channel 12 is sensed, a printer will space one line and skip to channel 1; a card punch will select punch pocket 1.

Cards can be punched only in EBCDIC mode.

Concatenating Sequential and Partitioned Data Sets

Two or more sequential or partitioned data sets can be automatically retrieved by the system and processed successively as a single data set. This reading technique is known as *concatenation.* A maximum of 255 data sets (16, if partitioned) can be concatenated, but they must be used only for input.

To save time when processing two consecutive data sets on a single volume, you specify LEAVE in your OPEN macro instruction. Concatenated data sets cannot be read backward.

When data sets are concatenated, the system treats the group as a single data set and only one data extent block (DEB) is constructed. Thus, it is important to consider the characteristics of the individual data sets being concatenated. Data sets with like characteristics are those that may be processed correctly using the same data control block (DCB), input/output block (IOB), and channel program. Any exception makes them unlike. Concatenated partitioned data sets are always treated as like and use the attributes of the first data set only. You must inform the system if unlike data sets are concatenated by modifying the DCBOFLGS field of the DCB. The indication must be made before the end of the current data set is reached. You must set bit 4 to 1 by using the instruction OI DCBOFLGS,X'08' as described in "Modifying the Data Control Block." If bit 4 of the DCBOFLGS field is 1, end–of–volume processing for each data set will issue a CLOSE for the data set just read and an OPEN for the next concatenated data set. This opening and closing procedure updates the fields in the DCB and, if necessary, builds a new IOB and a new channel program. If the buffer pool was obtained automatically by the Open routine, the procedure also frees the buffer pool and obtains a new one for the next concatenated data set. The procedure does not issue a FREEPOOL for the last concatenated data set. Unless you have some way of determining the characteristics of the next data set before it is opened, you should not reset the DCBOFLGS field to indicate like characteristics during processing.

When unlike data sets have been concatenated, you should not issue multiple input requests, that is, a series of READ or GET macro instructions, in your program. If you do, you will have to arrange some way to determine which requests have been completed and which must be reissued. In any case, the GET or READ macro instruction that detected the end of data set will have to be reissued. Figure 64 illustrates a possible routine for determining when a GET or READ must be reissued.

This restriction does not apply to like data sets since no open or close operation is necessary between data sets.

When the change from one data set to another is made, label exits are taken as required; automatic volume switching is also performed for multiple–volume data sets unless they are partitioned. If you are concatenating partitioned data sets on multiple volumes, all the volumes must be mounted before program execution. Your end–of–data–set (EODAD) routine is not entered until the last data set has been processed, except that for partitioned data sets, your EODAD routine receives control at the end of each member. At that time, you can process the next member or close the data set.

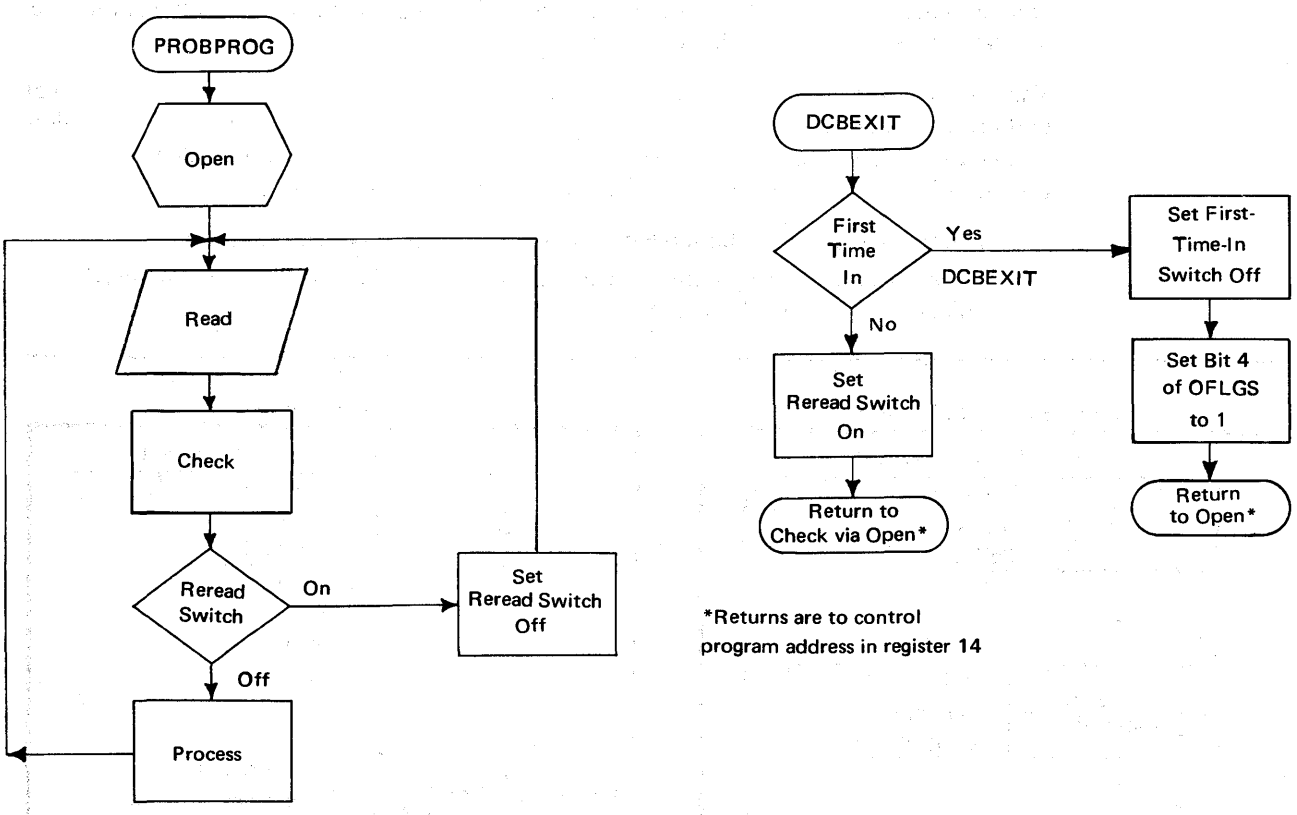


Figure 64. Reissuing a READ for Unlike Concatenated Data Sets

You process a concatenation of partitioned data sets the same way you process a single partitioned data set with one exception. You must use the FIND macro instruction to begin processing a member; you cannot use the POINT (or NOTE) macro instruction until after the FIND macro instruction has been issued. Example 13 shows how to process a single partitioned data set using FIND. If two members of different data sets in the concatenation have the same name, the FIND macro instruction determines the address of the first one in the concatenation. You would not be able to process the second one in the concatenation. The BLDL macro instruction provides the concatenation number of the data set to which the member belongs in the K field of the build list. See the section "BLDL–Construct a Directory Entry List" under "Processing a Partitioned Data Set" in Part 2 of this book.

Further discussion and examples of concatenated data sets are contained in *OS Job Control Language Reference.*

Cataloging Data Sets

The OS catalog is itself a data set residing on one or more direct–access volumes. It is organized into levels of indexes that connect the data–set names to corresponding volumes and data–set sequence numbers. For each level of qualification in the data–set name, there is an index group in the catalog.

The highest level of the catalog resides on the system–residence volume. The VTOC contains an entry for the DSCB defining the catalog and its highest–level index, the volume index. The lowest–level index contains the simple name of the data set and the number of the volume on which it resides.

The complete catalog can exist on the system–residence volume, or you can specify that parts of it be constructed on other volumes. Any volume containing part of the catalog is called *a control volume.* The use of control volumes allows data sets that are functionally related to be cataloged separately. The advantages include:

- Control volumes can be moved from one processing system to another.
- System–residence requirements can be reduced by placement of seldom–used indexes on a control volume.

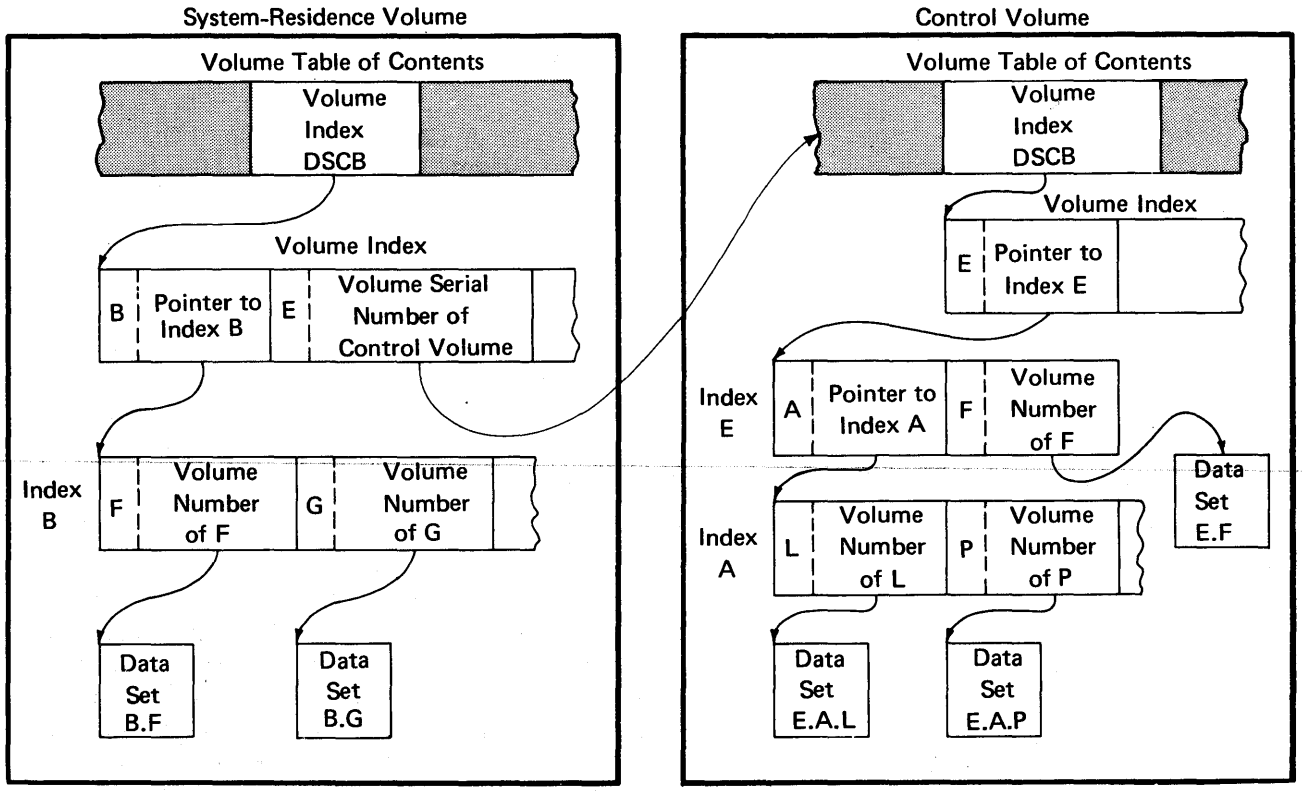


Figure 65. Catalog Structure on Two Volumes

For any given data set, only one level of control volume, other than the system–residence volume, can be used. Notice that in Figure 65, INDEX E, which is the highest–level index on the control volume, has an entry in both volume indexes.

The same type of cataloging is available for maintaining generation data groups. Cataloging each new generation data set with a unique name would be both inconvenient and inefficient. If you catalog individual data sets in a chronological collection by number, the entire collection can be stored under a single data set name.

Each update of the data set is called a generation; the number associated with it is called a generation number. A generation data group is the entire collection of chronologically related data sets that can be referred to by the same data set name. A particular generation can be referred to by either the absolute generation name or the relative generation number of the data set.

Absolute Generation Name: The operating system assigns each data set in the generation data group an absolute generation name in the form GggggVvv:

- gggg is an unsigned 4–digit decimal generation number.
- vv is an unsigned 2–digit decimal version number.

The generation number indicates how far removed the data set is from the original generation. The version number indicates how many times the associated generation has been replaced. Only the most recent version of a specific generation is retained.

Generation Increment: You can specify the increment by which the generation number is changed. For example, if you request a current generation G0013V04 and an increment of 2, the new generation would be assigned the absolute generation name G0015V00.

Version Increment: When you replace the same generation with a new version, it is your responsibility to assign the new, nonzero version number.

Concatenated Generations: By specifying only the data group name, you can request a concatenation of all existing data sets in the generation data group, starting with the most recent and ending with the oldest, with unit affinity to the most recent.

Relative Generation Number: Rather than request a data set by its absolute generation number, you can refer to it relative to the most recent generation, that is, DSNAME=dsname(n). Those immediately preceding the most recent are identified by *n* values of –1, –2, etc. You create new generations by referring to them as DSNAME=name(+1), name (+2), name (+3), etc. The last of these is cataloged as name (0) and the other generations in the catalog are adjusted accordingly at the end of the job.

Entering a Data Set Name in the Catalog

The catalog structure, including all levels of indexes, is initially created or modified by the utility program IEHPROGM. A data set name can then be entered if the proper index levels of the name exist.

For example, if a data set named A.B.C is to be cataloged, the volume index on the system–residence volume must have an index entry for index A, which must point to an index B. When the data set A.B.C is cataloged, C is entered into index B along with

the volume serial number of the volume where data set A.B.C resides. The cataloging request is entered as:

```
//ddname DD DSNAME=A.B.C,DISP=( ,CATLG )
```

Entering a Generation Data Group in the Catalog

A data set that is part of a generation data group is represented in the catalog by an additional level of index that contains an entry for each generation. The utility program IEHPROGM is used to create the index levels and to instruct the system in how the generations are to be maintained.

Controlling Confidential Data — Password Protection

In addition to the usual label protection that prevents opening of a data set without the correct data set name, the operating system provides data set security options that prevent unauthorized access to confidential data. Two levels of protection options are available. You specify these options in the LABEL field of a DD statement with the parameter PASSWORD or NOPWREAD.

- Password protection (specified by the PASSWORD parameter) makes a data set unavailable for all types of processing until a correct password is entered by the system operator.
- No–password–read protection (specified by the NOPWREAD parameter) makes a data set available for input without a password, but requires that the password be entered for output or delete operations.

If an incorrect password is entered twice, the job is terminated by the system.

You can request password protection when you create the data set by using the LABEL field of the DD statement in your JCL. The system sets the data set security byte either in the standard header label 1 as shown in *OS Tape Labels* or in the identifier data set control block (DSCB) as shown in *OS System Control Blocks*. Once you have requested security protection, you cannot remove it with JCL unless you recreate the data set and scratch the protected data set. You can add protection to an old data set or change it from NOPWREAD to PASSWORD if you open the data set for OUTPUT or OUTIN when you open it for the first time during a job step.

In addition to requesting password protection in your JCL, you must enter at least one record for each protected data set in a data set name PASSWORD that must be created on the system–residence volume. *OS DADSM Logic*, GY28–6607, contains a description of the record format for the PASSWORD data set. You should also request password protection for the PASSWORD data set itself to prevent both reading and writing without knowledge of the password.

For a data set on a direct–access device you can place the data set under protection at the same time that you enter its password in the PASSWORD data set. You can use the PROTECT macro instruction or the IEHPROGM utility program to add, change, or delete an entry in the PASSWORD data set; with either of these methods the system updates the DSCB of the data set to reflect its protected status. This provision eliminates the need for you to use JCL whenever you add, change, or remove security protection for a data set on a direct–access device. A description of how to maintain the PASSWORD data set, including the PROTECT macro instruction, is contained in *OS Data Management for System Programmers*. *OS Utilities* describes IEHPROGM.

APPENDIX A: DIRECT–ACCESS LABELS

Only standard label formats are used on direct–access volumes. Volume, data set, and optional user labels are used (see Figure 66). In the case of direct–access volumes, the data set label is the data set control block (DSCB).

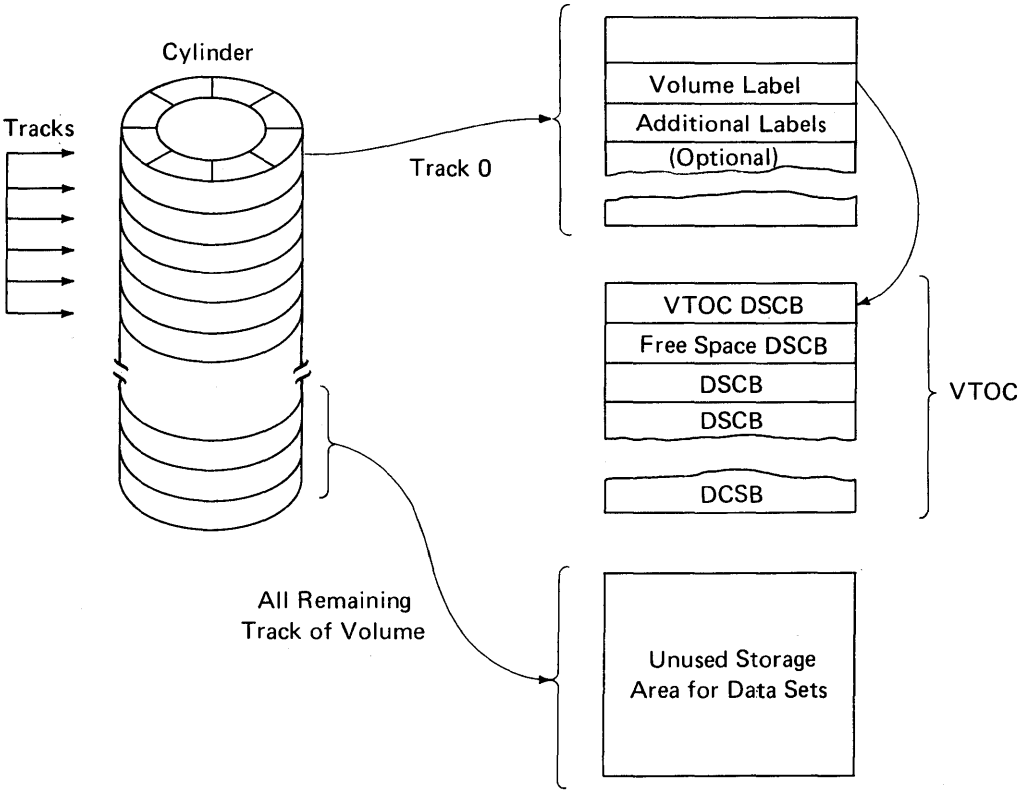


Figure 66. Direct–Access Labeling

Volume–Label Group

The volume–label group immediately follows the initial program loading (IPL) records on track 0 of cylinder 0 of the volume. It consists of the initial volume label plus a maximum of seven additional volume labels. The initial volume label identifies a volume and its owner, and is used to verify that the correct volume is mounted. It can also be used to prevent use of the volume by unauthorized programs. The additional labels are processed by an installation routine that is incorporated into the system.

The format of the direct–access volume label group is shown in Figure 67.

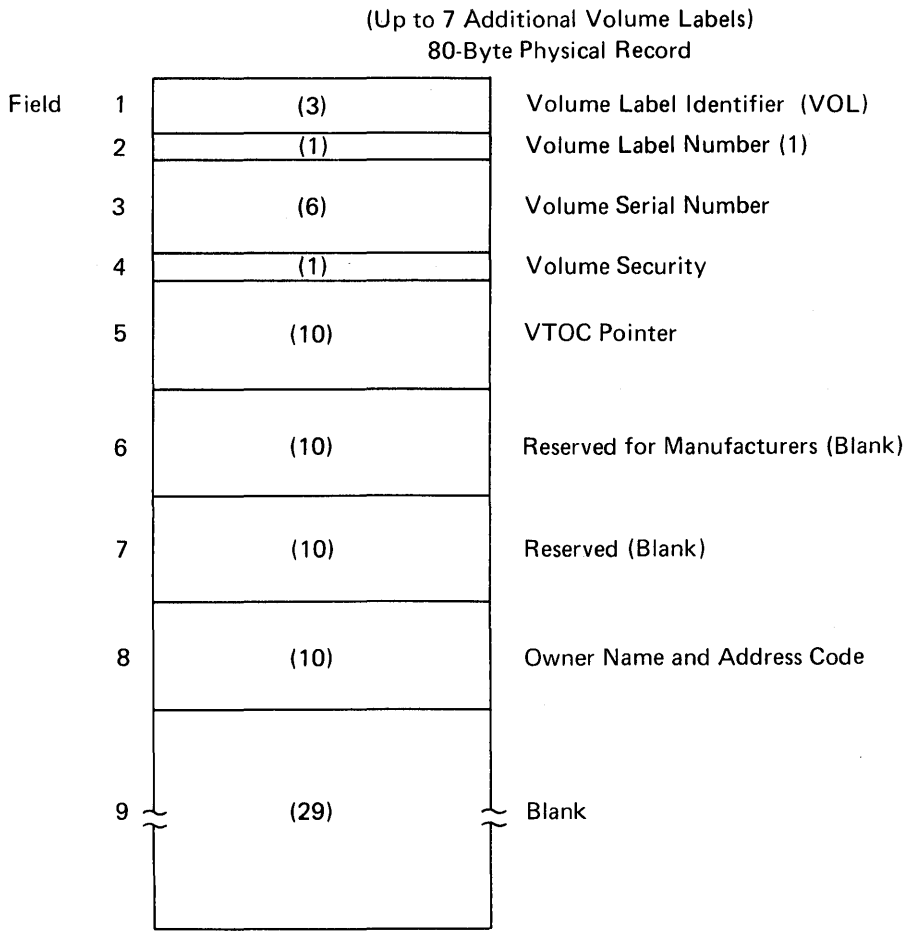


Figure 67. Initial Volume Label

Initial Volume Label Format

Volume Label Identifier (VOL): Field 1 contains the initial volume label.

Volume Label Number (1): Field 2 identifies the relative position of the volume label in a volume label group. It must be written as 1.

The operating system identifies an initial volume label when, in reading the initial record, it finds that the first 4 characters of the record are VOL1.

Volume Serial Number: Field 3 contains a unique identification code assigned when the volume enters the system. You can place the code on the external surface of the volume for visual identification. The code is normally numeric (000001–999999), but may be any 6 alphameric characters.

Volume Security: Field 4 is reserved for future use by installations that wish to provide security for volumes. It must be written as 0.

VTOC Pointer: Field 5 of direct–access volume label 1 contains the address of the VTOC.

Reserved for Manufacturers: Field 6 is reserved for future standardization purposes. Leave it blank.

Reserved: Field 7 is reserved for future developmental purposes. Leave it blank.

Owner Name and Address Code: Field 8 contains a unique identification of the owner of the volume.

All of the bytes in Field 9 are left blank.

Data Set Control Block (DSCB)

The system automatically constructs a DSCB when space is requested for a data set on a direct–access volume. Each data set on a direct–access volume has one or more DSCBs to describe its characteristics. The DSCB appears in the VTOC and contains operating–system data, device–dependent information, and data set characteristics, in addition to space allocation and other control information. There are seven kinds of DSCBs, each with a different purpose and a different format number. For an explanation of the seven kinds of DSCBs, see *OS System Control Blocks.*

User Label Groups

User header and trailer label groups can be included with data sets of physically sequential or direct organization. The labels in each group have the format shown in Figure 68.

Each group can include up to eight labels, but the space required for both groups must not be more than 1 track on a direct–access device. The current minimum track size allows a maximum of eight labels, including both header and trailer labels. Consequently, a program becomes device–dependent (among direct–access devices) when it creates more than eight labels.

If user labels are specified in the DD statement (LABEL=SUL), an additional track is normally allocated when the data set is created. No additional track is allocated when specific tracks are requested (SPACE=(ABSTR,...)), or when tracks allocated to another data set are requested (SUBALLOC=...). In either case, labels are written on the first track that is allocated.

User Header Label Group: The operating system writes these labels as directed by the processing program recording the data set. The first 4 characters of the user header label must be UHL1,..., UHL8; you can specify the remaining 76 characters. When the data set is read, the operating system makes the user header labels available to the problem program for processing.

User Trailer Label Group: These labels are recorded (and processed) as explained in the preceding text for user header labels, except that the first 4 characters must be UTL1,...., UTL8.

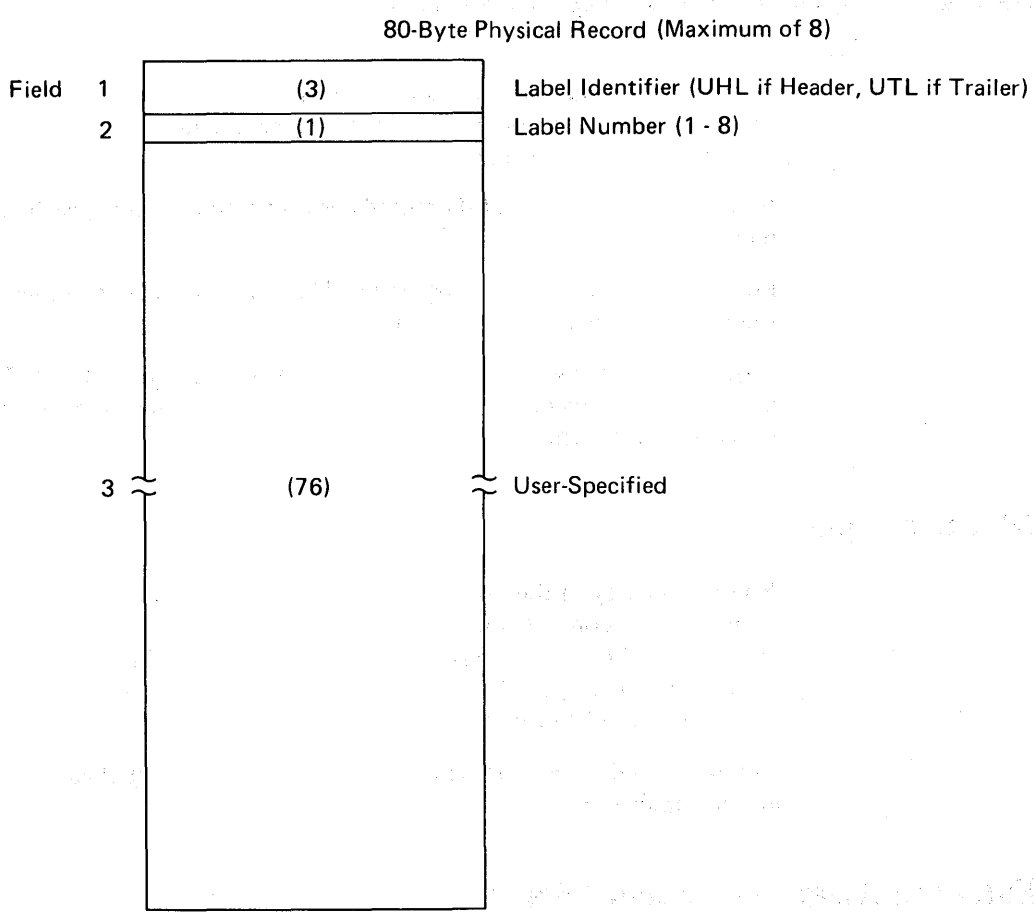


Figure 68. User Header and Trailer Labels

User Header and Trailer Label Format

Label Identifier: Field 1 indicates the kind of user header label. UHL indicates a user header label; UTL indicates a user trailer label.

Label Number: Field 2 identifies the relative position (1–8) of the label within the user label group.

User–Specified: Field 3 (76 bytes).

APPENDIX B: CONTROL CHARACTERS

As an optional feature, each logical record, in any record format, may include a control character. This control character is recognized and processed if a data set is being written to a printer or punch.

For format–F and format–U records, this character is the first byte of the logical record.

For format–V records, it must be the fifth byte of the logical record, immediately following the record descriptor word.

Two options are available. If either option is specified in the DCB, the character must appear in every record and other line spacing or stacker selection options also specified in the DCB are ignored.

Machine Code

You can specify in the DCB that the machine code control character has been placed in each logical record. If the record is to be written, the appropriate byte must contain the command code bit configuration specifying both the write and the desired carriage or stacker select operation. If the record is not to be written, the byte can specify any command other than write.

Command codes for specific devices are contained in publications describing the control units and devices.

Extended American National Standards Institute Code

In place of machine code, you can specify control characters defined by the American National Standards Institute, Inc. (ANSI). These characters must be represented in EBCDIC.

The extended American National Standards Institute (ANSI) code is as follows:

Code	Action Before Printing a Line
b	Space one line (blank code)
0	Space two lines
-	Space three lines
+	Suppress space
1	Skip to channel 1
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7
8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12
V	Select punch pocket 1
W	Select punch pocket 2

These control characters include those defined by ANSI FORTRAN. If any other character is specified, it is interpreted as 'b' or V, depending on the device being used; no error indication is returned.

APPENDIX C: SPECIAL PROGRAMMING CONSIDERATIONS FOR THE 3505 CARD READER AND THE 3525 CARD PUNCH

Using BSAM or QSAM, you can read cards on the 3505 Card Reader, or you can read, punch, interpret punch (punch and interpret the punches), or print cards on the 3525 Card Punch. There are no new programming considerations when you read or punch cards unless you read using read column eliminate (RCE) or optical mark read (OMR), which are explained in the last section of this appendix. The following three sections will help you write programs to interpret punch, to print, or to do more than one operation on each card during the execution of your program.

3525 Interpret Punch

You can interpret punch by specifying FUNC=I in the DCB. LRECL must be 80 or 81 if you use a control character. The first 64 characters that you punch in a card are printed (interpreted) on line 1 of the card; the last 16 characters are printed right–justified on line 3 of the card. One output macro instruction is all you need to both punch and interpret the punches on a card.

3525 Print

You can print two lines of data or as many as 25 lines of data on a card. If you specify FUNC=WT in the DCB, you can print on lines 1 and 3 only (two–line print). If you specify FUNC=W, you can print as many as 25 lines on a single card (multiline print). You can print data that is identical to or different from the data which is punched on the card.

You effect stacker selection for cards when printing only by using the STACK operand of the DCB macro instruction.

You can control line positioning and card feeding by using the CNTRL macro instruction or by using the control characters explained in "Appendix B: Control Characters." If you use ANSI control characters, abnormal termination results when you try to space and print beyond line 3 (two–line print) or beyond line 25 (multiline print) or when you try to suppress space and print on any line. You feed the next card by skipping to a channel with a number equal to or less than the channel number on the present card. Figure 69 shows the correspondence between print line number and channel numbers.

If you do not control printing by using the CNTRL macro instruction or by using control characters, print lines are single–spaced and cards are fed automatically.

3525 Associated Data Sets

You can perform more than one operation on each card during the execution of your program. The data used for each operation is considered as an individual data set and each data set must have a separate DCB. For example, if you want to read a card, punch additional data into the card, and print data on the card, you must specify three DCBs. These three data sets are called *associated data sets.* You indicate that they are

associated by coding the proper subparameters for FUNC in each DCB. In this example you code FUNC=RPW. See *OS Data Management Macro Instructions* for the other FUNC subparameters you would code for other associated data sets.

You must also indicate that the data sets are associated with the same device by using the unit affinity parameter of the DD statement. On the first DD statement for an associated data set you would code UNIT=3525 or UNIT=unitaddress; on the DD statements for the other associated data sets you would code UNIT=AFF=READ, where READ is the name of the first DD statement. The following JCL examples show how you would code the UNIT parameter using device type and the DCB parameter of the DD statement.

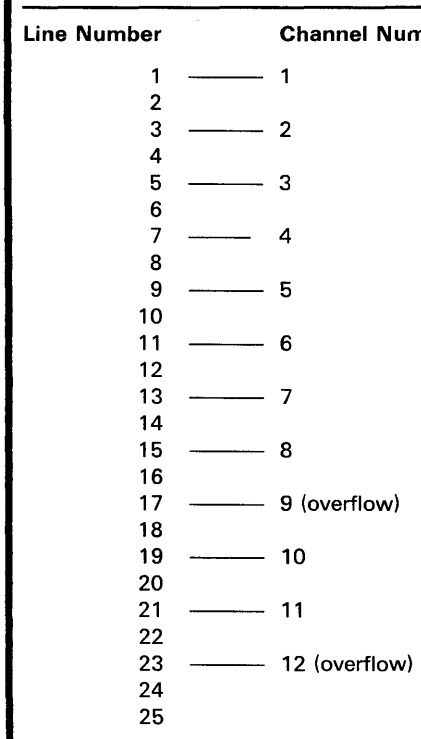


Figure 69. Correspondence Between Print Line Numbers and Channel Numbers

Read and punch associated data sets:

```
//READ   DD   UNIT=3525,DCB=(FUNC=RP)
//PUNCH  DD   UNIT=AFF=READ,DCB=(FUNC=RP)
```

Read, punch, and print associated data sets:

```
//READ   DD   UNIT=3525,DCB=(FUNC=RPW)
//PUNCH  DD   UNIT=AFF=READ,DCB=(FUNC=RPW)
//PRINT  DD   UNIT=AFF=READ,DCB=(FUNC=RPWX)
```

Read and print associated data sets:

```
//READ   DD   UNIT=3525,DCB=(FUNC=RW)
//PRINT  DD   UNIT=AFF=READ,DCB=(FUNC=RWX)
```

Punch and print associated data sets:

```
//PUNCH  DD   UNIT=3525,DCB=(FUNC=PW)
//PRINT  DD   UNIT=AFF=PUNCH,DCB=(FUNC=PWX)
```

Only four combinations of operations are allowed for associated data sets. You can read and punch; read, punch, and print; read and print; or punch and print. The following restrictions apply to associated data sets:

- I/O operations on a single card must be completed in the sequence read, punch, print. Two reads in succession or two punches in succession cause abnormal termination if performed on the read and punch data sets or the read, punch, and print data sets. A print operation can be omitted or repeated, but the first line on a card cannot be printed until the card has been punched or, if the card is not to be punched, until the card has been read.
- You can use either BSAM or QSAM to process associated data sets, but the same access method must be used on all the associated data sets in the program. Otherwise abnormal termination occurs.
- The FUNC parameter must be coded in the DCB or the DD statement for each associated data set. See *OS Data Management Macro Instructions* for a complete description of the FUNC parameter.
- Associated data sets cannot be allocated to SYSIN or SYSOUT. You must request the specific device type or unit address in the UNIT parameter of the DD statement. See *OS Job Control Language Reference* and the examples above for more information on the DD statement and the UNIT parameter.
- BUFNO=1 must be specified to read or punch an associated data set.
- When one of the associated data sets is to be punched, stacker selection can occur only with the punch data set. You can accomplish this by using control characters or the STACK operand of the DCB macro instruction. For the read and print associated data sets where no punch data set is used, stacker selection can be specified only with the read read data set through the CNTRL macro instruction.
- To prevent punching card columns that already contain data, you request the data protection option in the FUNC parameter of the DCB macro instruction. This option applies only to the read, punch, and print associated data sets and the read and punch associated data sets. *OS Data Management Macro Instructions* explains how to request the data protection option in the FUNC parameter of the DCB macro instruction.
- An attempt to feed a card during printing causes abnormal termination. The next card feeds automatically when you request the next read or punch.

Opening Associated Data Sets

Associated data sets can be opened in any order, but you cannot process one associated data set unless all associated data sets are open. If an I/O operation is requested for a data set when one or more of its associated data sets is not open, abnormal termination results.

Closing Associated Data Sets

Associated data sets can be closed in any order, but once a data set is closed, I/O operations cannot be requested for any of its associated data sets. If such an operation is requested, abnormal termination results.

When you close the associated data set that causes card feeds, the CLOSE macro instruction causes a feed command to be issued. This ensures that your last data card is moved from the card transport to the stacker. If you do not close the data set that causes card feeds before your job terminates, your last data card will remain in the card transport. Figure 70 shows which operation causes a card feed for 3525 data sets and associated data sets.

Data Set	Operation Causing a Card Feed
Read	Read
Punch	Punch
Print	Print
Interpret Punch	Punch
Read and Punch	Read
Read, Punch, and Print	Read
Read and Print	Read
Punch and Print	Punch

Figure 70. Operations That Cause Card Feed for a 3525 Card Punch

If you use a data delimiter card for the input data set, your program must check for it and branch to your EODAD routine. If the end–of–file condition is sensed by the device, the system branches to your EODAD routine and causes a card feed. After your program reads a data delimiter card, do not attempt to punch or print on it; if you do, you will cause the first card of the following job to be lost. Your data delimiter card remains in the card transport until the following job causes a card feed or until a nonprocess runout is performed by the operator.

To prevent the loss of the last data card when data sets are closed, remember the following restrictions:

- All associated data sets must be closed before termination of the job step without intervening I/O operations for any of the associated data sets.
- If any data set is reopened, the appropriate associations must be reestablished.
- If the data set was read in read column eliminate (RCE) mode, a card feed will be issued to reset the mode to read 80 columns instead of the normal card feed that is issued when the data set is closed.

Optical Mark Read (3505 only) and Read Column Eliminate (3505 and 3525)

If you specify MODE=O for optical mark read (OMR) or MODE=R for read column eliminate (RCE) in the DCB, you must provide a format descriptor card as the first card of your data deck. The format descriptor card specifies the columns from which optical marks are to be read or the columns which are to be eliminated. Abnormal termination results if you do not supply a format descriptor card. If you use checkpoint/restart, the format of the OMR or RCE data set must be reestablished when the job is restarted.

Format Descriptor Card: The word FORMAT must be coded starting in column 2 of the first card of the data deck, followed by a blank and the parameters that specify the columns to be read in OMR mode or RCE mode. The remaining columns are read in the normal punched mode.

If columns 1, 3, 5, 7, 9, 70, 72, 74, 76, 78, and 80 are to be read in OMR mode, code FORMAT (1,9),(70,80). A maximum of 40 columns of OMR data can be read from each card.

If columns 20 through 30 and 52 through 76 are not to be read (RCE mode), code FORMAT (20,30),(52,76).

Continuation cards can be coded if necessary and are coded like macro instruction continuation cards.

OMR or RCE is in effect only while the data set that specifies it is open.

OMR Data Records: The following rules apply to coding an OMR record.

- Mark fields must be separated by at least one blank (not a punch or mark).
- Mark and punch fields must be separated by at least one blank (not a punch or mark).
- Mark fields in odd columns and mark fields in even columns must be separated by at least two blanks (not punches or marks).
- Mark or punch fields may begin in any column, so long as the coding conforms to the first three rules.

These rules and their application to an 80–column card are represented in Figure 71.

Although OMR data is physically located on the card in alternating columns, the data is compressed in the channel. The blank following an optical mark is not transferred to the input buffer. See Figure 71 for the format of the OMR data as it appears in the channel and input buffer.

When a marginal mark, weak mark, or poor erasure is detected, the column's data is replaced with X'3F' in EBCDIC or with X'3F3F' in column binary mode. X'3F' is also placed in column 80 for EBCDIC and column 160 for column binary. You are responsible for checking for OMR reading errors.

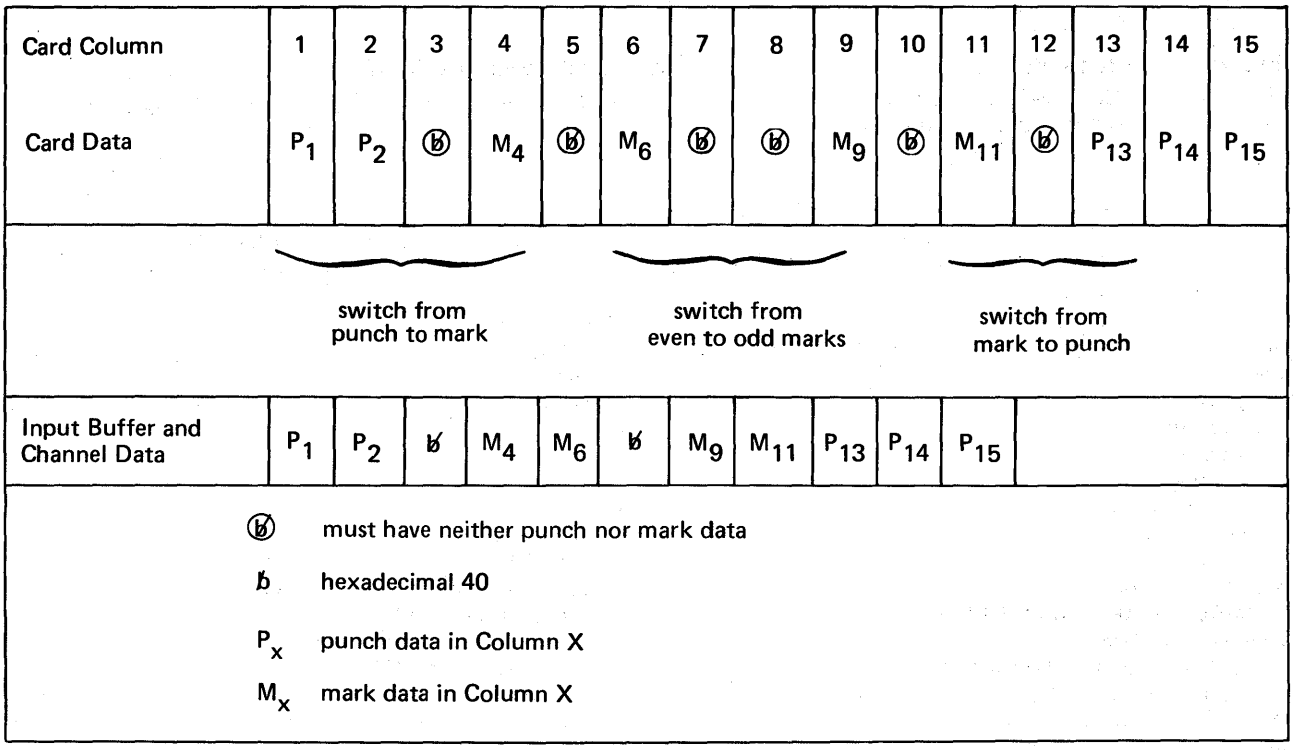


Figure 71. OMR Coding Rules

INDEX

Indexes to Systems Reference Library publications are consolidated in *OS Master Index to Reference Manuals,* GC28–6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the *Master Index.*

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

Order Number GC26-3746-1

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

READER'S COMMENT FORM

OS Data Management Services Guide — Order Number GC26-3746-1

Your comments about this publication will help us to produce better publications for your use. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes ☐

No ☐

Name _____

Job Title _____

Address _____

_____ Zip _____

No postage necessary if mailed in the U S A

Order Number GC26-3746-1

YOUR COMMENTS, PLEASE . . .

This publication is one of a series which serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. D78

fold fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)