IBM

MVS/Extended Architecture Data Administration Guide

Licensed Program

Order Number
GC26-4140-2

Data Facility Product
5665-XA2

Version 2
Release 3.0

IBM

MVS/Extended Architecture Data Administration Guide

Licensed Program

Order Number
GC26-4140-2

Data Facility Product
5665-XA2

Version 2
Release 3.0

Third Edition (June 1987)

This is a major revision of, and makes obsolete, GC26-4140-1.

This edition applies to Version 2 Release 3.0 of MVS/Extended Architecture Data Facility Product, Licensed Program 5665-XA2, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Changes" following the preface. Specific changes are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent publication of the page affected. Editorial changes that have no technical significance are not noted.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead.

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. If you request publications from the address given below, your order will be delayed because publications are not stocked there.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This book is intended for system programmers who use the IBM data management access methods—other than VSAM (virtual storage access method)—to process data sets. This book does not cover such specialized applications as time sharing option (TSO), graphics, teleprocessing, optical character readers, optical reader–sorters, and magnetic character readers. These specialized applications are described in separate publications listed in *IBM System/370 and 4300 Processors Bibliography*, GC20-0001.

To learn about VSAM or to write programs that create and process VSAM data sets, see:

- *MVS/Extended Architecture Catalog Administration Guide*, GC26-4138, which describes how to create master and user catalogs
- *MVS/Extended Architecture VSAM Administration Guide*, GC26-4151, which describes how to create VSAM data sets
- *MVS/Extended Architecture Integrated Catalog Administration: Access Method Services Reference*, GC26-4135, and *MVS/Extended Architecture VSAM Catalog Administration: Access Method Services Reference*, GC26-4136, which describe the access method services commands used to manipulate VSAM data sets
- *MVS/Extended Architecture VSAM Administration: Macro Instruction Reference*, GC26-4152, which describes how to code the macro instructions required with VSAM data sets

Organization

This publication has 15 chapters and 4 appendixes:

- Chapter 1, "Introduction to Data Administration" on page 1, provides an overview of data set processing, including a description of four different access methods, and a discussion of data set identification.
- Chapter 2, "Data Set Storage" on page 5, discusses the characteristics of data sets that are stored on direct access and magnetic tape devices.
- Chapter 3, "Record Formats" on page 13, discusses the considerations for creating and processing records within the various kinds of record formats.

- Chapter 4, "Selecting an Access Method" on page 33, gives an overview of basic and sequential access methods in data management, and compares the functions and performance of each.

- Chapter 5, "Specifying a Data Control Block and Initializing Data Sets" on page 39, discusses how to specify a DCB and how to open and close data sets. This chapter also discusses managing buffer pools and handling data set volumes.

- Chapter 6, "Accessing Records in Data Sets" on page 59, discusses how to use GET and PUT or READ and WRITE macros to access data records. This chapter also discusses analyzing input and output errors.

- Chapter 7, "DCB Exit Routines" on page 73, discusses user-written exit routines and the parameter lists they use.

- Chapter 8, "Spooling and Scheduling Data Sets" on page 75, discusses how to route data through the input/output streams of the job entry subsystem (JES).

- Chapter 9, "Processing a Sequential Data Set" on page 79, discusses managing sequential data sets and buffers.

- Chapter 10, "Processing a Partitioned Data Set" on page 101, discusses the advantages and restrictions of partitioned data sets.

- Chapter 11, "Processing a Direct Data Set" on page 119, discusses the tasks required to process BDAM data sets.

- Chapter 12, "Processing an Indexed Sequential Data Set" on page 129, discusses the organization of and the access techniques for ISAM data sets.

- Chapter 13, "Generation Data Groups" on page 163, discusses the reasons for using generation data groups and how to specify them.

- Chapter 14, "I/O Device Control Macros" on page 171, discusses the macro instructions that control input and output devices.

- Chapter 15, "Protecting Data" on page 175, discusses password and Resource Access Control Facility (RACF) protection of non-VSAM data sets.

- Appendix A, "Direct Access Labels" on page 179, discusses the standard label formats used on direct access volumes.

- Appendix B, "Control Characters" on page 183, discusses the use of an optional control character to control card punches and printers.

- Appendix C, "Allocating Space on Direct Access Volumes" on page 187, discusses methods of estimating capacities and space requirements on direct access devices.

- Appendix D, "ISO/ANSI/FIPS Record Control Word and Segment Control Word" on page 193, discusses the translation of ISO/ANSI/FIPS record control words and ISO/ANSI/FIPS segment control words.

Prerequisite Knowledge

To use this book efficiently, you should be familiar with:

- Assembler language
- Job control language

Required Publications

You should be familiar with the information presented in the following publications:

- *Assembler H Version 2 Application Programming: Guide*, SC26-4036
- *Assembler H Version 2 Application Programming: Language Reference*, GC26-4037
- *MVS/Extended Architecture JCL User's Guide*, GC28-1351
- *MVS/Extended Architecture JCL Reference*, GC28-1352

Related Publications

Within the text, references are made to the publications listed below:

Short Title	Publication Title	Order Number
Assembler H V2 Application Programming: Guide	*Assembler H Version 2 Application Programming: Guide*	SC26-4036
Assembler H V2 Application Programming: Language Reference	*Assembler H Version 2 Application Programming: Language Reference*	GC26-4037
Data Administration: Macro Instruction Reference	*MVS/Extended Architecture Data Administration: Macro Instruction Reference*	GC26-4141

Short Title	Publication Title	Order Number
Debugging Handbook	*MVS/Extended Architecture Debugging Handbook*, Volumes 1 through 5	LC28-1164[1] LC28-1165 LC28-1166 LC28-1167 LC28-1168
Device Support Facilities User's Guide and Reference	*Device Support Facilities User's Guide and Reference*	GC35-0033
Data Facility Product: Customization	*MVS/Extended Architecture Data Facility Product: Version 2 Customization*	GC26-4267
IBM 3262 Model 5 Printer Product Description	*IBM 3262 Model 5 Printer Product Description*	GA24-3936
IBM 3800 Printing Subsystem Programmer's Guide	*IBM 3800 Printing Subsystem Programmer's Guide*	GC26-3846
IBM 3800 Printing Subsystem Programmer's Guide	*IBM 3800 Printing Subsystem Models 3 and 8 Programmer's Guide*	SH35-0061
IBM 3890 Document Processor Machine and Programming Description	*IBM 3890 Document Processor Machine and Programming Description*	GA24-3612
IBM 4245 Printer Model 1 Component Description and Operator's Guide	*IBM 4245 Printer Model 1 Component Description and Operator's Guide*	GA33-1541
IBM 4248 Printer Model 1 Description	*IBM 4248 Printer Model 1 Description*	GA24-3927
Initialization and Tuning	*MVS/Extended Architecture System Programming Library: Initialization and Tuning*	GC28-1149
JCL User's Guide	*MVS/Extended Architecture JCL User's Guide*	GC28-1351
JCL Reference	*MVS/Extended Architecture JCL Reference*	GC28-1352
Magnetic Tape Labels and File Structure Administration	*MVS/Extended Architecture Magnetic Tape Labels and File Structure Administration*	GC26-4145

Note:

1 All five volumes may be ordered under one order number, LBOF-1015.

Short Title	Publication Title	Order Number
Open/Close/EOV Logic	*MVS/Extended Architecture Open/Close/EOV Logic*	LY26-3966
OS/VS IBM 3886 Optical Character Reader Model 1 Reference	*OS/VS IBM 3886 Optical Character Reader Model 1 Reference*	GC24-5101
OS/VS Mass Storage System (MSS) Planning Guide	*OS/VS Mass Storage System (MSS) Planning Guide*	GC35-0011
OS/VS Mass Storage System (MSS) Services: General Information	*OS/VS Mass Storage System (MSS) Services: General Information*	GC35-0016
OS/VS Mass Storage System (MSS) Extensions Services: Reference	*OS/VS Mass Storage System (MSS) Extensions Services: Reference*	SH35-0036
Programming Support for the IBM 3505 Card Reader and the IBM 3525 Card Punch	*Programming Support for the IBM 3505 Card Reader and the IBM 3525 Card Punch*	GC21-5097
RACF General Information Manual	*Resource Access Control Facility (RACF): General Information Manual*	GC28-0722
Service Aids	*MVS/Extended Architecture System Programming Library: Service Aids*	GC28-1159
Supervisor Services and Macro Instructions	*MVS/Extended Architecture System Programming Library: Supervisor Services and Macro Instructions*	GC28-1154
System Codes	*MVS/Extended Architecture Message Library: System Codes*	GC28-1157
System–Data Administration	*MVS/Extended Architecture System–Data Administration*	GC26-4149
System Generation	*MVS/Extended Architecture Installation: System Generation*	GC26-4148
System Macros and Facilities	*MVS/Extended Architecture System Programming Library: Macros and Facilities,* Volumes 1 and 2	GC28-1150 and GC28-1151

Short Title	Publication Title	Order Number
System Messages	*MVS/Extended Architecture Message Library: System Messages*, Volumes 1 and 2	GC28-1376 and GC28-1377
Utilities	*MVS/Extended Architecture Data Administration: Utilities*	GC26-4150

Summary of Changes

Release 3.0, June 1987

New Programming Support

This release provides virtual storage constraint relief by allowing VSAM control blocks to be obtained above 16M. A new long form parameter list allows for 31-bit addresses. Changes have been made to Chapter 5, "Specifying a Data Control Block and Initializing Data Sets" to document the new long form parameter list. The default is the standard form parameter list with 24-bit addresses.

A new DCB exit list code, X'13', has been added to RDJFCB to retrieve allocation information. Changes have been made to Chapter 7, "DCB Exit Routines" to document the new DCB exist list code.

Customization Restructure

Detailed information on user exits has been moved from Chapter 7, "DCB Exit Routines" on page 73 to *Data Facility Product: Customization.*

Service Changes

Minor technical changes have been made.

Release 2.0, June 1986

RACF Protection

New information has been added in Chapter 15, "Protecting Data" on page 175 on RACF protection for non-VSAM data sets and tape volumes. Information has also been added on erasing RACF-protected DASD data sets.

Some technical changes have been made. These changes are reflected with change bars.

Release 1.0, April 1985

New Device Support

Modifications have been made to the SETPRT macro to support the IBM 4248, 3262 Model 5, and 4245 printers.

Note: The IBM 3262 Model 5 printer is afforded the same support as that provided for the IBM 4248 printer. However, the use of an IBM 3262 Model 5 printer dictates that entries exist in the IBM 4248 printer UCS Image Table for the IBM 3262 Model 5 printer band(s)/alias(es) used on the host system. These image table entries must be generated by the user as part of the IBM 3262 Model 5 installation procedure.

New Programming Support

Information has been added to "RACF Protection for Non-VSAM Data Sets" on page 176 that tells you how to overwrite RACF-protected DASD data sets.

Other support information added consists of a write validity check option enhancement for the IBM 3480.

Version 2 Publications

The Preface includes order numbers for Version 2.

Contents

Figures

Chapter 1. Introduction to Data Administration

Data administration is the process of systematically and effectively organizing, identifying, storing, cataloging, and retrieving all the information (including programs) that your installation uses.

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

A *data set* is a collection of logically related data records that are stored on a volume and that may be classified according to installation needs. For example, a sales department could classify its data by geographic area, by individual salesperson, or by any other logical plan. A user can request data from a direct access volume or a tape volume.

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

Data administration provides:

- Allocation of space on direct access volumes.
- Automatic retrieval of cataloged data sets by name alone.

Overview of Data Set Processing

Input/output routines in the operating system schedule and control all data transfer operations between virtual and auxiliary storage. These routines can:

- Read data
- Write data
- Translate data from ISCII/ASCII (International Standard Code for Information Interchange and American National Standard Code for Information Interchange) to EBCDIC (Extended Binary Coded Decimal Interchange Code) and the reverse
- Block and unblock records

- Overlap reading, writing, and processing operations
- Read and verify volume and data set labels
- Write data set labels
- Position and reposition volumes automatically
- Detect I/O errors and correct them when possible
- Provide exits to user-written error and label routines

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

You can organize your data sets in one of four ways:

- *Sequential*: Records are organized 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. You must use the sequential data set organization for all magnetic tape devices, but it is optional on direct access devices. Punched cards and printed output must also be 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 and sequential access to any record.
- *Direct*: 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 called *libraries*.

Requests for input/output operations on data sets through macro instructions use 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 macros. Also, you may choose a data organization according to the access techniques and processing capabilities available.

The code generated by the macros for both techniques is optionally re-enterable, depending on the form in which parameters are expressed.

Besides the access methods provided by the operating system, an elementary access technique called *execute channel program* (EXCP) is also provided. To use this

technique, you must establish your own system for organizing, storing, and retrieving data. The primary advantage of EXCP is the complete flexibility it allows you in using the computer directly.

An important feature of data administration is that much of the detailed information needed to store and retrieve data, such as device type, buffer processing technique, and length 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 the queued and basic access techniques for processing sequential data sets. To some extent, you can determine the degree of device independence. Many useful device-dependent features are available as part of certain macro instructions; achieving device independence requires some selectivity in their use.

Identifying Data Sets

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 macros, 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 help 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. The first name is called the high-level qualifier, the last is the low-level qualifier.

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 that

specifies the particular data set to be used is reserved by a DCB macro when your program is assembled.

Chapter 2. Data Set Storage

The operating system provides 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 a reel of magnetic tape, a disk pack, or a drum.

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 where 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, and 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 where 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.

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; and so forth. The same data set names can be used repeatedly with no need to keep track of the volume serial numbers used. For more information, see "Relative Generation Number" on page 164.

Direct Access Volumes

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 find any record without extensive searching. Thus, records can be stored and retrieved either directly or sequentially.

Direct access volumes are used to store executable programs, including the operating system itself. Direct access storage (sometimes called DASD 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 that is stored at track 0 of cylinder 0. You may specify as many as seven additional labels, located following 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. The format 0 DSCB describes an unused (available) record in the VTOC. *System—Data Administration* describes format 1 through format 6 DSCBs and their purposes. *System—Data Administration* also describes the structure of the VTOC.

Each direct access volume is initialized by a utility program before being used on the system. The initialization program generates the volume label and builds the table of contents. For additional information on direct access labels, see Appendix A, "Direct Access Labels" on page 179.

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.

Track Characteristics

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.

Track Format

Information is recorded on all direct access volumes in a standard format. Besides device data, each track contains a track descriptor record (capacity record or record 0) and data records.

Figure 1 shows that 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.

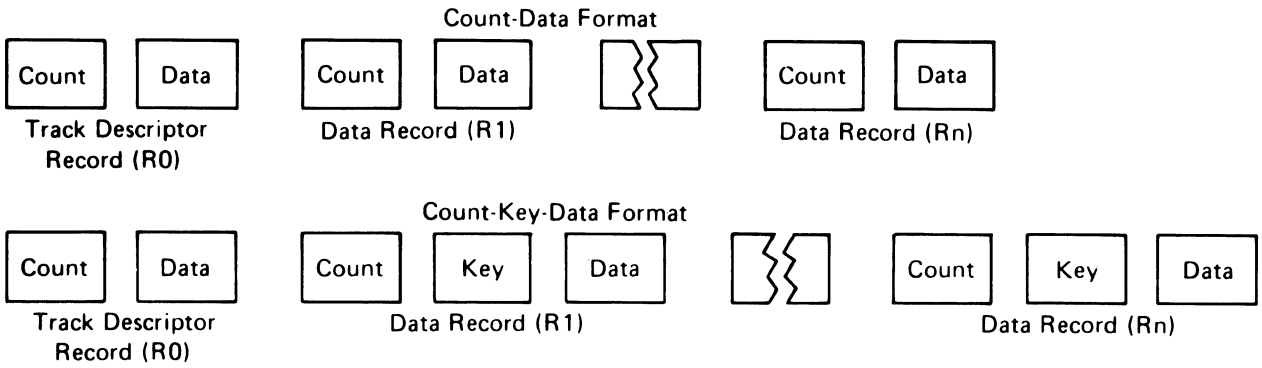


Figure 1. Direct Access Volume Track Formats

Besides 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 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 DCB parameter of the DD statement, or the DCB macro associated with the data set. If the overflow 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 where 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.

Actual and Relative 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 DEB 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 shows the position of the block relative to the first block of the data set. Allocation of noncontiguous 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 noncontiguous 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.

Magnetic Tape Volumes

Because data sets on magnetic tape devices must be organized sequentially, the operating system does not require space allocation procedures comparable to that 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. The operating system positions a volume with IBM standard labels, ISO/ANSI/FIPS 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* is the data set sequence number, and *yy* is the volume sequence number for the data set. If you specify volume serial numbers for unlabeled volumes where 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 ensures that data sets residing on different volumes are not cataloged or passed under identical volume serial numbers. Retrieving such data sets can give unpredictable errors.

Each data set and data set label group must be followed by a tapemark. Tapemarks cannot exist within a data set. When the operating system creates a tape with standard labels or no labels, all tapemarks are automatically written. Two tapemarks follow the last trailer label group on a standard-label volume. On an unlabeled volume, the two tapemarks appear after the last data set.

When the operating system creates data sets with nonstandard labels, no tapemarks are written. If you want the operating system to retrieve a data set, you must supply the tapemarks in your routine that creates the nonstandard-label volume. Otherwise, tapemarks are not required after nonstandard labels, because positioning of the tape volumes must be handled by installation routines.

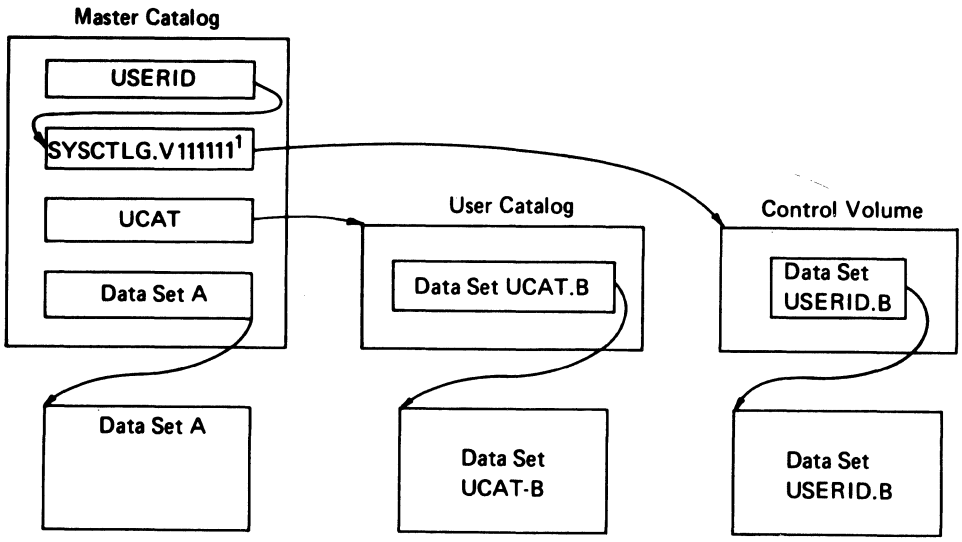
For more information about magnetic tape volume labels, see *Magnetic Tape Labels and File Structure*. For more information about nonstandard labels, see *Data Facility Product: Customization*.

The data on magnetic tape volumes can be in either EBCDIC or ISCII/ASCII. ISCII/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 ISCII/ASCII and of the International Organization for Standardization (ISO), American National Standards Institute (ANSI), and Federal Information Processing Standard (FIPS) tape labels is such that data management can translate records on input tapes in ISCII/ASCII into EBCDIC for internal processing and translate the EBCDIC into ISCII/ASCII for output. Records on such input tapes may be sorted into ISCII/ASCII collating sequence.

Cataloging Data Sets

The operating system has a catalog structure consisting of a master catalog, user catalogs, and, optionally, OS CVOLs. Figure 2 shows the catalog structure.



[1] 111111 is the volume serial of the OS CVOL.

Figure 2. Catalog Structure

There is one master catalog on each system. It is required and contains entries for system data sets. It is also the VSAM or integrated catalog facility master catalog and does not have to be on the system residence volume. The master catalog contains a pointer to each user catalog. Both VSAM and non-VSAM data sets can be cataloged in a user catalog.

Non-VSAM data sets can be cataloged on OS CVOLs (SYSCTLG data sets). The master catalog contains a pointer to each OS CVOL. Data sets can be cataloged, uncataloged, or recataloged. (For more information on using OS CVOLs, see *Catalog Administration Guide.*) If a data set is not cataloged in the master catalog, the first name of a qualified data set name indicates the user catalog or OS CVOL in which it is cataloged. A user catalog can also be connected to the system as a job catalog or a step catalog.

Permanent Mass Storage System (MSS) data sets should be cataloged to allow efficient use of the mass storage volume control (MSVC) functions. For information on MSVC, see *OS/VS Mass Storage System (MSS) Services: General Information.*

Entering a Data Set Name in the Catalog

The data set name of a non-VSAM data set can be entered in a master or user catalog through (1) job control language (DISP parameter), (2) access method services (DEFINE command), or (3) catalog management macro instructions (CATALOG and CAMLST). A non-VSAM data set name can be entered in an OS CVOL through JCL or the catalog management macros. VSAM data sets can only be cataloged by using access method services.

Access method services is also used to establish aliases for data set names and to connect user catalogs and OS CVOLs to the master catalog. For information on how to use the access method services commands, see *Access Method Services Reference*. For information on how to use the catalog management macro instructions, see *Catalog Administration Guide* and *System Data Administration*.

Data set names cannot be cataloged in an OS CVOL if a name is already cataloged whose levels match the highest or higher levels of the specified name. For example, the qualified name A.B.C.D cannot be cataloged if the name A.B or A.B.C is already cataloged, but the name A.B.C.D can be cataloged if AB.C or A.B.C.E is cataloged. This restriction is *not* true for data sets cataloged in an integrated catalog facility or VSAM catalog.

Chapter 3. Record Formats

The *record* is the basic unit of information used by a processing program and can be anything from a single character to a mass of information collected by a particular business transaction, or measurements recorded at a given point in an experiment. A collection of logically related records makes up a data set. Most data processing consists of reading, manipulating, and writing individual records.

Blocking is the process of grouping records before they are written on a volume. A block consists of one or more logical records written between consecutive interrecord gaps (IRGs). Blocking conserves storage space on a volume by reducing the number of IRGs in the data set and increases processing efficiency by reducing the number of input/output operations required to process the data set.

Records are stored in one of four formats: fixed-length (format-F), variable-length for data in EBCDIC (format-V) or for data to be translated to or from ISCII/ASCII (format-D), or undefined-length (format-U).

Before selecting a record format, you should consider:

- The data type (for example, EBCDIC) your program will receive and the type of output it will produce
- The input/output devices that will contain the data set
- The access method you will use to read and write the records

You identify your record format selection in the data control block using the options in the DCB macro, the DD statement, or the data set label.

ISO/ANSI/FIPS tape records are written in format-F, format-D, format-S or format-U with the restrictions noted under "Fixed-Length Records, ISO/ANSI/FIPS Tapes" on page 15, "ISO/ANSI/FIPS Variable-Length Records—Format D" on page 23, and "Undefined-Length Records" on page 27.

Note: Data can only be in format-U for ISO/ANSI Version 1 tapes (ISO 1001-1969 and ANSI X3.27-1969).

When data management reads records from ISO/ANSI/FIPS tapes, it translates the records into EBCDIC. When data management writes records onto ISO/ANSI/FIPS tapes, it translates the records into ISCII/ASCII characters. 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: Translation routines supplied by the system will convert to ASCII 7-bit code, as explained in *Magnetic Tape Labels and File Structure*. When the character to be translated contains a bit in the high order position, the 7-bit translation does not produce an equivalent character. Instead, it produces a substitute character to note the loss in translation. This means, for example, that random binary data (such as a dump) cannot be recorded in ISO/ANSI/FIPS 7-bit code.

Fixed-Length Records

The size of fixed-length (format-F) records, shown in Figure 3, 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 (except for card readers) on blocked or unblocked format-F records. Allowances are made for truncated blocks.

Format-F records are shown in Figure 3. The optional control character (a), used for stacker selection or carriage control, may be included in each record to be printed or punched.

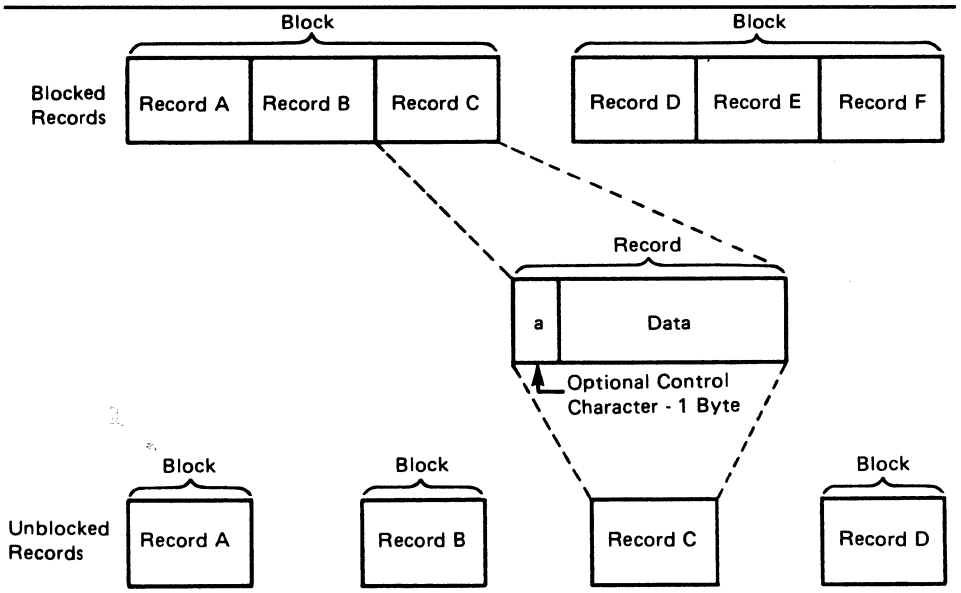


Figure 3. Fixed-Length Records

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 contains the same number of blocks.
- Every track except the last is filled to capacity as determined by the track capacity formula established for the device. (These formulas are presented in Appendix C, "Allocating Space on Direct Access Volumes" on page 187.)
- The data set organization is physical sequential. A member of a partitioned data set cannot be specified.

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 standard-format data set (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. Reading an extended data set with this condition will result in a premature end of data condition when the truncated block is read, giving the appearance that the blocks following this truncated block do not exist. This type of data set on magnetic tape should not be read backward, because the data set would begin with a truncated block. Consequently, you probably won't 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 use the track overflow feature.

Standard format should not be used to read records from a data set that was created using a RECFM other than standard, because other record formats may not create the precise format required by standard.

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

Fixed-Length Records, ISO/ANSI/FIPS Tapes

For ISO/ANSI/FIPS tapes, format-F records are the same as described above, with three exceptions:

- Control characters, if present, must be ISO/ANSI/FIPS control characters. For more information about control characters, see Appendix B, "Control Characters" on page 183.
- Record blocks can contain block prefixes.

  The block prefix can vary 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 do not

have access to 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 using QSAM to output DB or DBS records and BUFOFF=0 is specified, the value of the BUFL operand, if specified, must be increased by 4. If BUFL is not specified, then the BLKSIZE operand must be increased by 4. This allows for a 4-byte QSAM internal processing area to be included when the system acquires the buffers. These 4 bytes will not become part of the user's block.

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, as for all the data records for ISO/ANSI/FIPS tapes, can only contain EBCDIC characters that correspond to the 128, seven-bit ASCII characters. Thus, you must avoid using data types such as binary, packed decimal, and floating point that cannot always be translated into ISCII/ASCII. (See the **Note** in Chapter 3, "Record Formats" on page 13.)

Figure 4 on page 17 shows the format of fixed-length records for ISO/ANSI/FIPS tapes and where control characters and block prefixes are positioned if they exist.

- The GET routine tests each record (except the first) for all circumflex characters (X'5E'). If a record completely filled with circumflex characters is detected, the end-of-block (EOB) routine is called to get the next block. A fixed-length record must not consist of only circumflex characters. This restriction is necessary because circumflex characters are used to pad out a block of records when fewer than the maximum number of records are included in a block, and the block is not truncated.

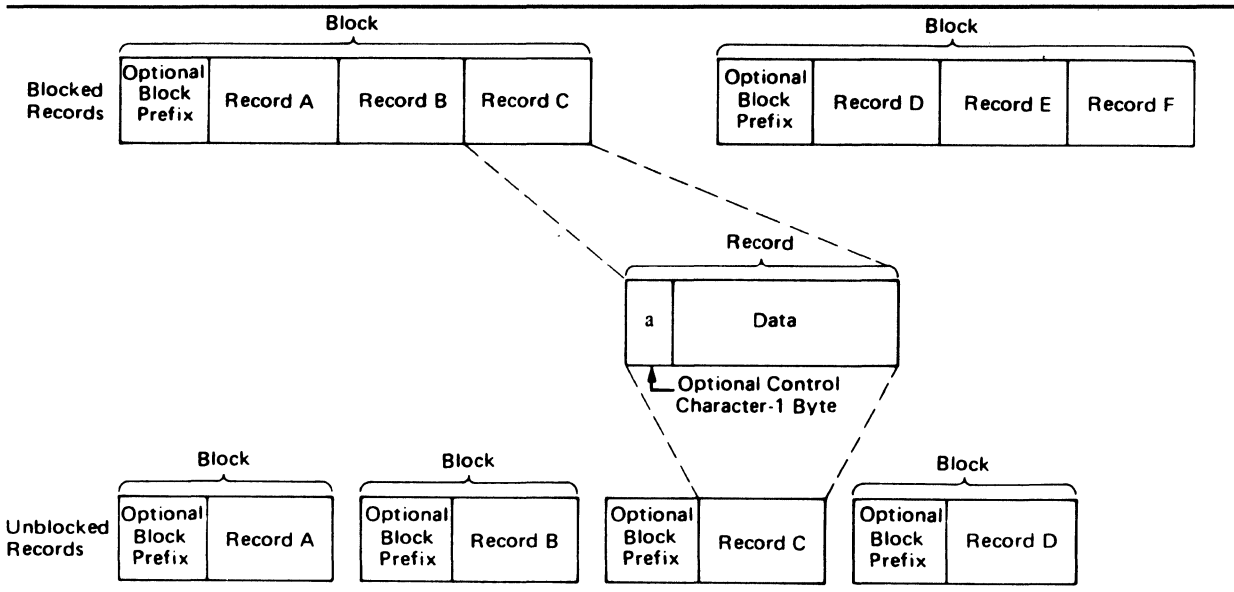


Figure 4. Fixed-Length Records for ISO/ANSI/FIPS Tapes

Variable-Length Records

The variable-length record formats are format-V and format-D. They can also be spanned (format-VS or -DS), blocked (format-VB or -DB), or both (format-VBS and -DBS). Format-D, -DS, and -DBS records are used for ISO/ANSI/FIPS tape data sets. Figure 5 on page 18 shows blocked and unblocked variable-length (format-V) records without spanning.

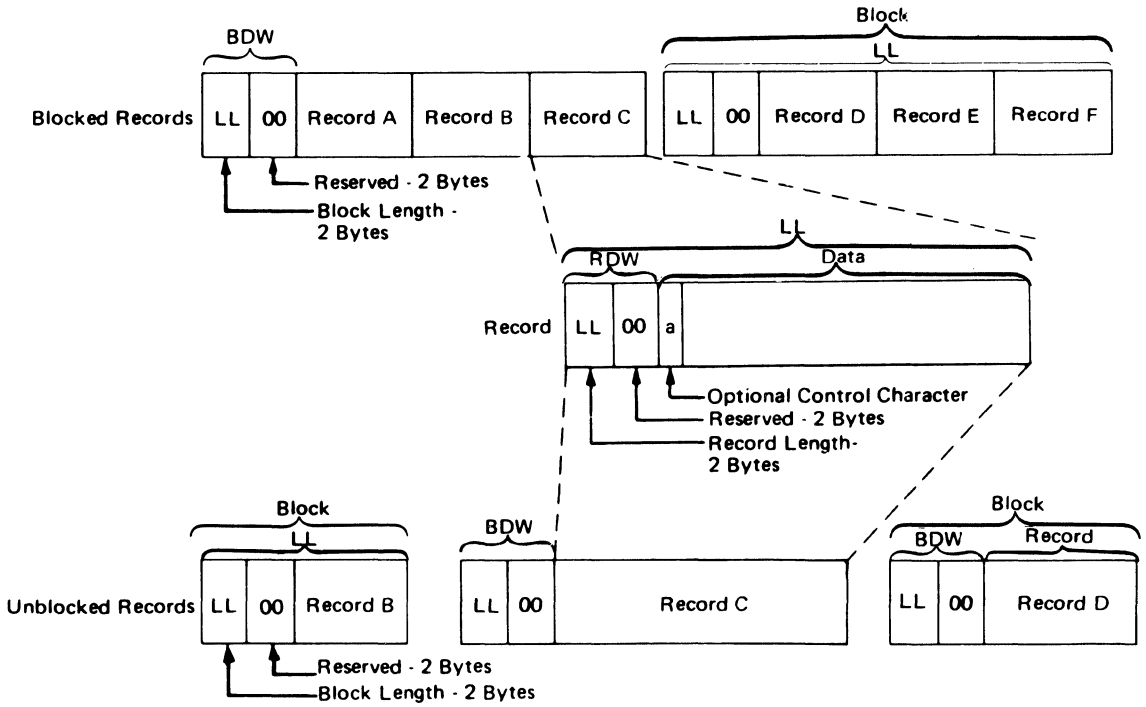


Figure 5. Nonspanned, Format-V Records

Variable-Length Records—Format-V

Format V provides for variable-length records and variable-length record segments, each of which describes its own characteristics, and for variable-length blocks of such records or record segments. Except when variable-length track overflow records are specified for volumes on devices with the rotational position sensing feature, the control program performs length checking of the block and uses the record or segment length information in blocking and unblocking. 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 32760 bytes or, when you are using WRITE with tape, from 18 to 32760. The third and fourth bytes are reserved for possible 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.

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 32756. All bits of the third and fourth bytes must be 0, because 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" on page 93). 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 (a) may be specified as the fifth byte of each record and must be followed by at least one byte of data (the length in the RDW, in this case, would be 6). The first byte of data is a table reference character if OPTCD=J has been specified. The RDW, the optional control character, and the optional table reference character are not punched or printed.

Spanned Format-VS Records (Sequential Access Method)

Figure 6 on page 20 shows how the spanning feature of the queued and basic sequential access methods lets you 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.

When spanning is specified for blocked records, the system tries to fill all blocks. For unblocked records, a record larger than block size is split and written in two or more blocks, each block containing only one record or record segment. Thus the block size 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" on page 93) by QSAM. For blocked records, 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 block size or record size parameters, the record segmenting will change.

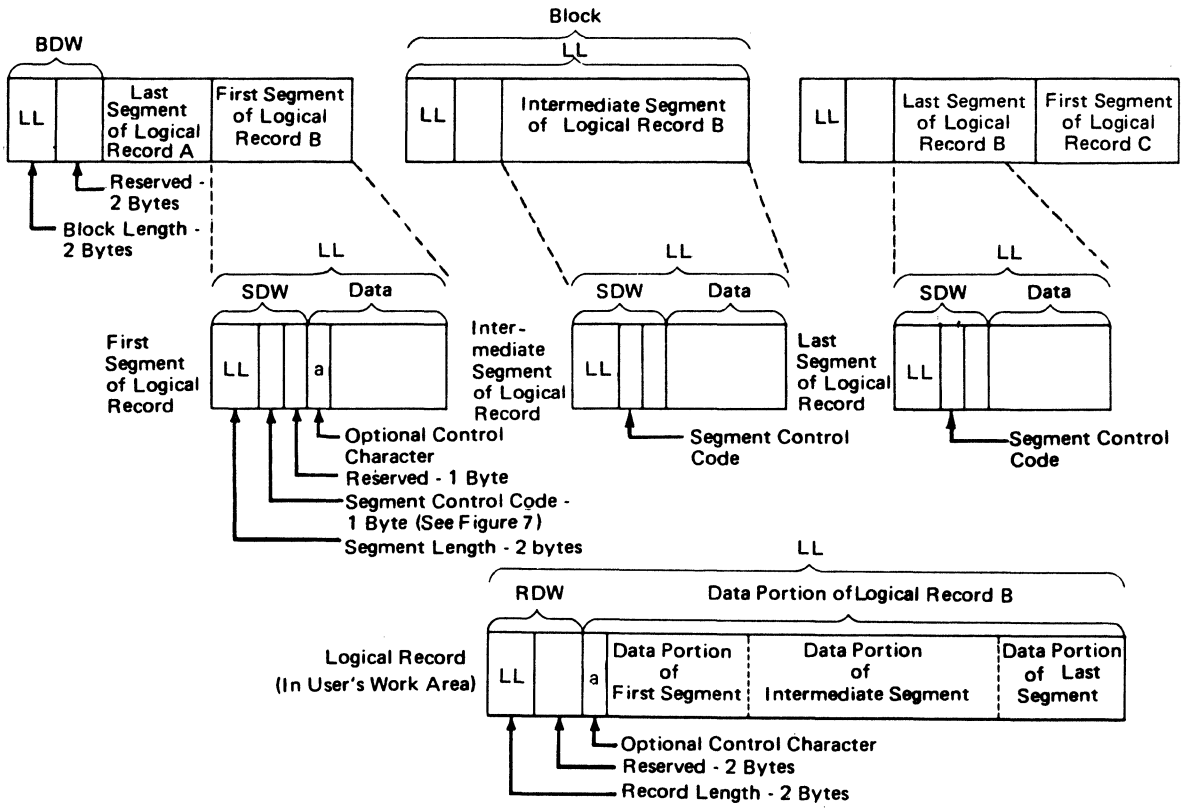


Figure 6. Spanned Format-VS Records (Sequential Access Method)

Considerations for Processing Spanned Record Data Sets

When spanned records span volumes, reading errors may occur when using QSAM if a volume that begins with a middle or last segment is mounted first or if an FEOV macro instruction is issued followed by another GET. QSAM cannot begin reading from the middle of the record. The errors include duplicate records, program checks in the user's program, and invalid input from the spanned record data set.

When QSAM opens a spanned record data set in UPDAT mode, it uses logical record interface (LRI) to assemble all segments of the spanned record into a single, logical input record and to disassemble a single logical record into multiple segments for output data blocks. A record area must be provided by using the BUILDRCD macro instruction or by specifying BFTEK=A in the DCB.

Note: When you specify BFTEK=A, the Open routine provides a record area equal to the LRECL specification, which should be the maximum length in bytes. (An LRECL=0 is invalid.)

If you issue the FEOV macro when reading a data set that spans volumes, or if a spanned multivolume data set is opened to other than the first volume, make sure that each volume begins with the first (or only) segment of a logical record. Input routines cannot begin reading in the middle of a logical record.

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 32756 bytes or, when you are using WRITE with tape, from 18 to 32756 bytes. The third byte of the SDW contains the segment control code that 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 7. The remaining bits of the third byte and all of the fourth byte are reserved for possible 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 or last segment

Figure 7. Segment Control Codes

The SDW for the first segment replaces the RDW for the record after the record has been segmented. You or the operating system can build the SDW, depending on which access method 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 (that is, to one segment) in your work area. The logical record length is passed to you through the DCBLRECL field of the 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 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. (BFTEK=A or the BUILDRCD macro cannot be specified when logical records exceed 32760 bytes. To process logical records that exceed 32760 bytes, you must use locate mode and specify LRECL=X in your DCB macro.)

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 that unblocked records are being processed and the block size must be equivalent to the length of one print line or one card. Records that span blocks are written one segment at a time.

Note: Spanned variable-length records cannot be specified for a SYSIN data set.

Null Segments

A 1 in bit position 0 of the SDW indicates a null segment. A null segment means that there are no more segments in the block. Bits 1 to 7 of the SDW and the remainder of the block must be binary zeros. A null segment is not an end-of-logical-record delimiter. (You do not have to be concerned about null segments unless you have created a data set using null segments.)

Spanned Variable-Length Records (Basic Direct Access Method)

The spanning feature of the basic direct access method (BDAM) lets you create and process variable-length unblocked logical records that span tracks. The feature also lets you pack tracks with variable-length records by splitting the records into segments. Figure 8 shows how these segments can then be written onto more than one track.

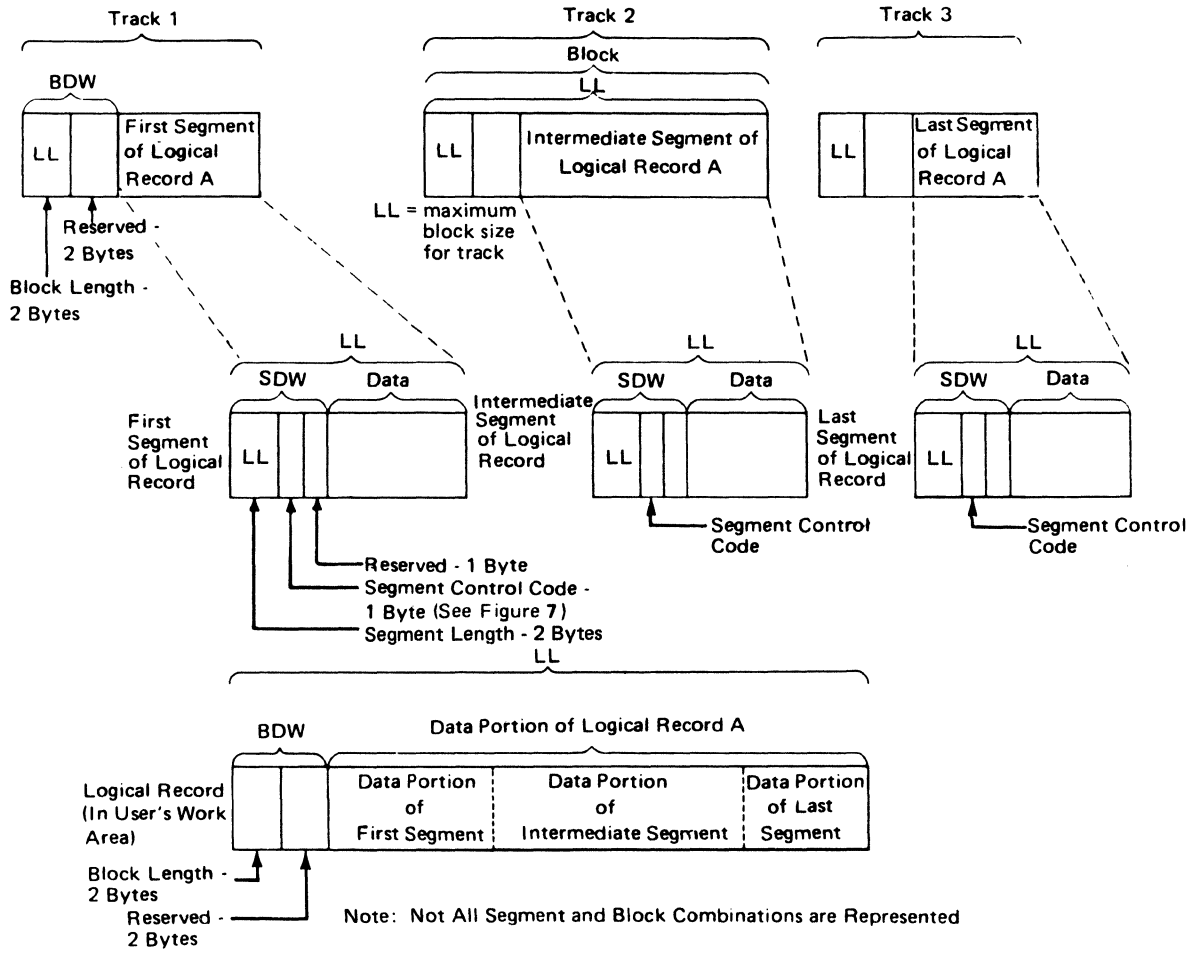


Figure 8. Spanned Format-V Records for BDAM Data Sets

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 track capacity. Furthermore, segmenting records allows a

record to span several tracks, with each segment of the record on a different track. However, because 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.

ISO/ANSI/FIPS Variable-Length Records—Format D

For ISO/ANSI/FIPS tapes, nonspanned variable-length records are format-D records. ISO/ANSI/FIPS records are the same as format-V records, with the following exceptions:

- *Block prefix*—A record block can contain a block prefix. To specify a block prefix, code the BUFOFF operand in the DCB macro. The block prefix can vary in length from 0 to 99 bytes but its length must remain constant for all records in the data set being processed. For blocked records, the block prefix precedes the RDW for first or only logical record in each block. For unblocked records, the block prefix precedes the RDW for each logical record.

  To specify that the block prefix is to be treated as a BDW by data management for format-D or -DS 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. The maximum length of a format-D or -DS, BUFOFF=L block is 9999, because the length (stated in binary by the user) is translated to a 4-byte ASCII character decimal field on the ISO/ANSI/FIPS tape when the block is written. It is converted back to a 2-byte length field in binary followed by two bytes of zeros when the block is read. 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. If you are using chained scheduling to read blocked DB or DBS records, you cannot code BUFOFF=*absolute expression* in the DCB. Instead, BUFOFF=L is required, because the access method needs binary RDWs and valid block lengths to unblock the records.

  When you use QSAM, you cannot read the block prefix into your record area 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.

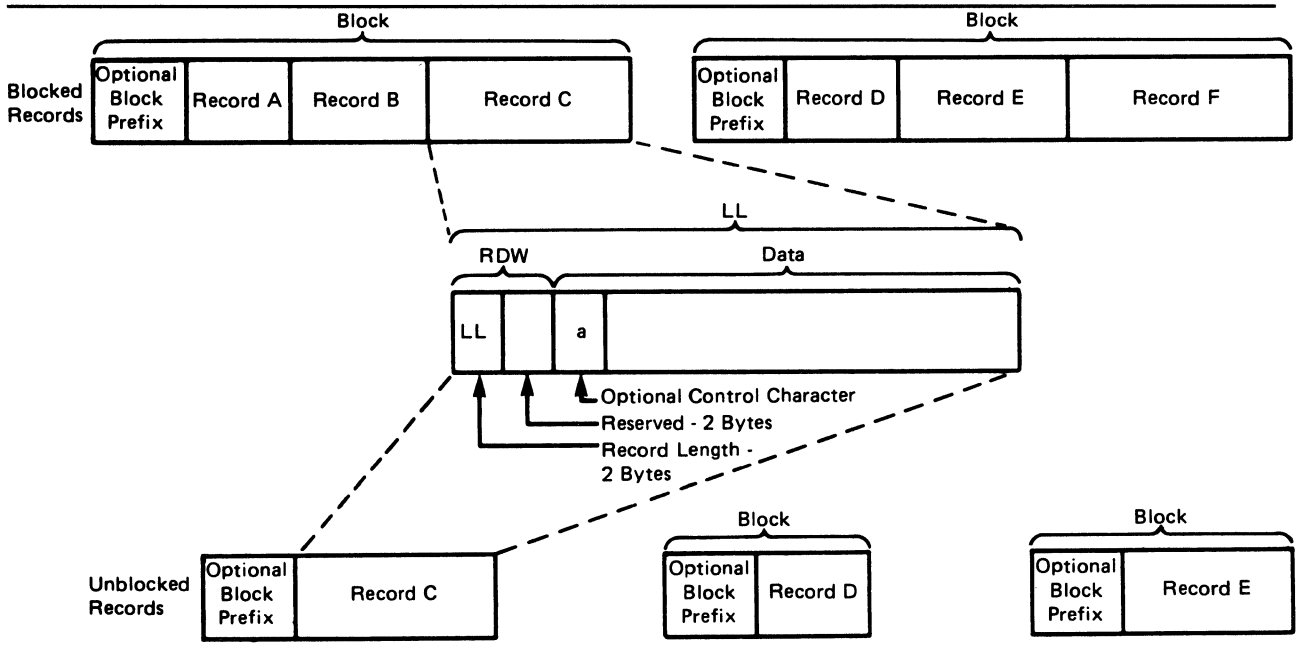
  The block prefix can only contain EBCDIC characters that correspond to the 128, seven-bit ASCII characters. Thus, you must avoid using data types, such as binary, packed decimal, and floating point, that cannot always be translated into ISCII/ASCII. (See the **Note** in Chapter 3, "Record Formats" on page 13.) For DB and DBS 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.

- *Block size*—Version 3 tapes have a maximum block size of 2048. This limit may be overridden by a label validation installation exit.

If you create variable-length blocks that are shorter than 18 bytes, data management pads each one to 18 bytes when the blocks are written onto an ISO/ANSI/FIPS tape. The padding character used is the ISCII/ASCII circumflex character.

- *Control characters*—Control characters, if present, must be ISO/ANSI control characters. For more information about control characters, see Appendix B, "Control Characters" on page 183.

Figure 9 shows the format of nonspanned variable-length records for ISO/ANSI/FIPS tapes, where the record descriptor word (RDW) is located, and where block prefixes and control characters must be placed when they are used.



Note: Block prefixes on output records must be 4-bytes long.
Figure 9. Nonspanned Format-D Records for ISO/ANSI/FIPS Tapes

ISO/ANSI/FIPS Variable-Length Spanned Records—Format-DS or Format-DBS

For ISO/ANSI/FIPS tapes, variable-length spanned records must be specified in the DCB RECFM parameter as DCB RECFM=DS or DBS. Format-DS and -DBS records are similar to format-VS or -VBS records with the following exceptions:

- *Segment descriptor word (SDW)*—There is an additional byte preceding each SDW for DS/DBS records. This additional byte is required for conversion of the SDW from IBM to ISO/ANSI/FIPS format, because the ISO/ANSI SDW (called a segment control word) is five bytes long. Otherwise, the SDW for DS/DBS records is the same as the SDW for VS/VBS records. The SDW LL count excludes the additional byte. (See "Processing Considerations for DS and DBS Records" on page 25.)

- *Extended logical record interface (XLRI)*—DS/DBS records may be processed using XLRI. (See "Processing Considerations for DS and DBS Records" on page 25.)

- The exceptions previously noted ("ISO/ANSI/FIPS Variable-Length Records—Format D" on page 23) for format-D records still apply.

Figure 10 on page 26 shows what spanned variable-length records for ISO/ANSI/FIPS tapes look like when you are using IBM access methods. The figure shows the segment descriptor word (SDW), where the record descriptor word (RDW) is located, and where block prefixes must be placed when they are used. If you are not using IBM access methods, see Appendix D, "ISO/ANSI/FIPS Record Control Word and Segment Control Word" on page 193, for a description of ISO/ANSI/FIPS record control words and segment control words.

Processing Considerations for DS and DBS Records

When using QSAM, the same application used to process VS/VBS tape files can be used to process DS/DBS tape files. However, you must ensure that ISO/ANSI/FIPS requirements such as block size limitation, tape device, and restriction to EBCDIC characters that correspond to the 128, seven-bit ASCII characters are met. The SCW/SDW conversion and buffer positioning is handled by the GET/PUT routines.

When using BSAM to process a DS/DBS tape file, you must allow for an additional byte at the beginning of each SDW. The SDW LL must exclude the additional byte. On input, you must ignore the unused byte preceding each SDW. On output, you must allocate the additional byte for each SDW.

SDW Conversion: Sequential access method end-of-block (EOB) routines perform conversion between ISO/ANSI/FIPS segment control word (SCW) format and IBM segment descriptor word (SDW) format for both QSAM and BSAM processing. On output, the binary SDW LL value (provided by you when using BSAM and by the access method when using QSAM), is increased by 1 for the extra byte and converted to four ISO/ANSI/FIPS numeric characters. Because the binary SDW LL value will result in four numeric characters, the binary value must not be greater than 9998. The fifth character is used to designate which segment type (complete logical record, first segment, last segment, or intermediate segment) is being processed.

On input, the four numeric characters designating the segment length are converted to two binary SDW LL bytes and decreased by one for the unused byte. The ISO/ANSI/FIPS segment control character maps to the DS/DBS SDW control flags. This conversion leaves an unused byte at the beginning of each SDW. It is set to X'00'. For more detail on this process, see Appendix D, "ISO/ANSI/FIPS Record Control Word and Segment Control Word" on page 193.

XLRI Mode: The extended logical record interface (XLRI) may be used with DS/DBS files to communicate LRECL values over 32760. (XLRI is supported only in QSAM locate mode for ISO/ANSI/FIPS tapes.) XLRI should be used for any case where the logical record will exceed 32760 bytes. Using the LRECL=X for ISO/ANSI/FIPS causes an 013-DC abend.

To use XLRI, specify LRECL=0K or LRECL=nK in the DCB macro. Specifying

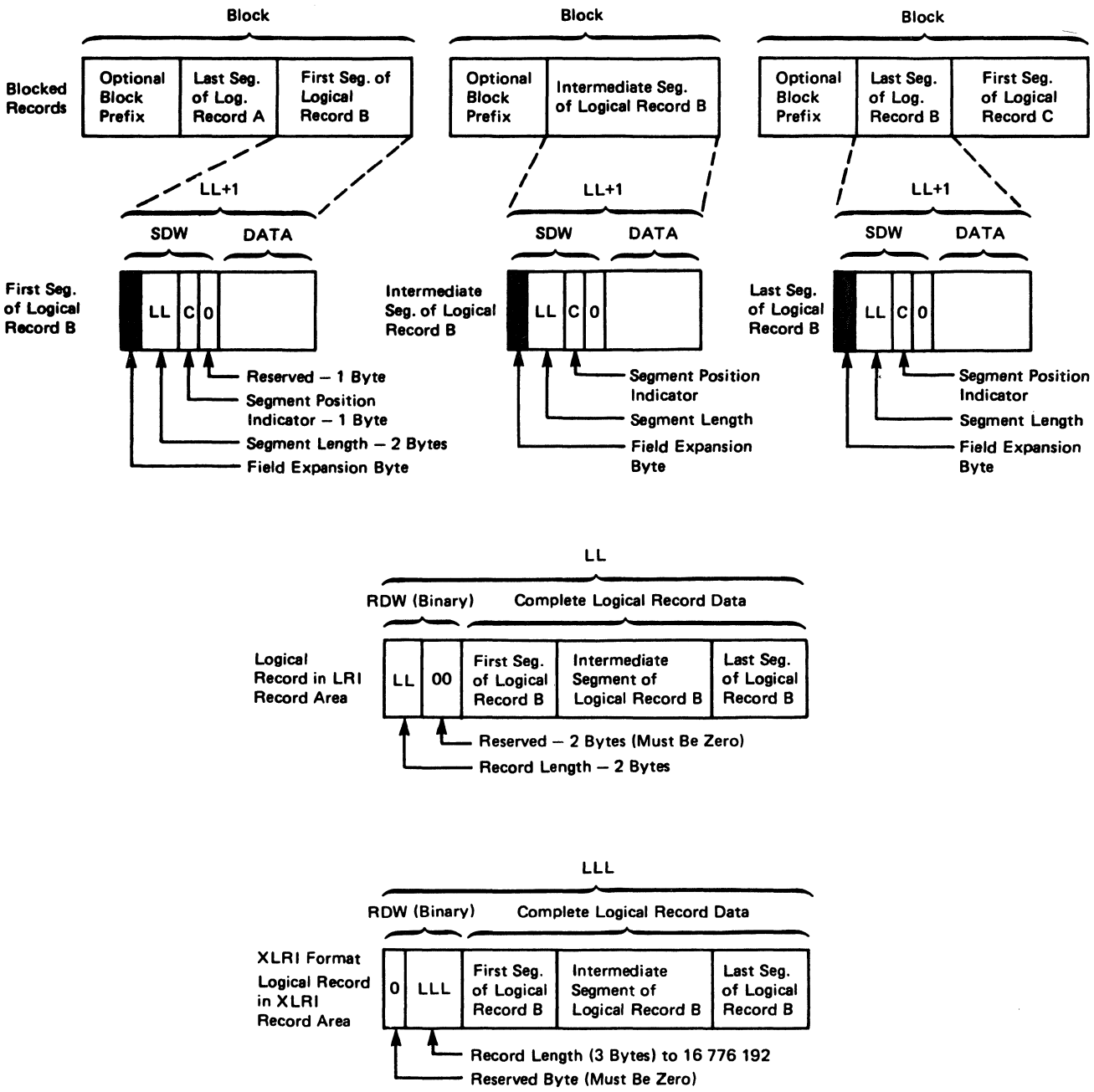


Figure 10. Spanned Variable-Length (Format-DS) Records for ISO/ANSI/FIPS Tapes

DCBLRECL with the K suffix sets the DCBBFTK bit that indicates that LRECL is coded in K units and that the DCB is to be processed in XLRI mode.

LRECL=0K in the DCB macro specifies that the LRECL value will come from the file label or JCL. When LRECL is from the label, the file must be opened as an input file. The label (HDR2) value for LRECL will be converted to kilobytes and rounded up when XLRI is in effect. When the ISO/ANSI/FIPS label value for LRECL is 00000 to show that the maximum record length may be greater than 99999, the LRECL=nK must be used in the JCL or in the DCB to specify the maximum record length.

The LRECL from JCL can be expressed in absolute form or with the K notation. Absolute values, permissible only from 5 to 32760, will be converted to kilobytes by rounding up to an integral multiple of 1024 when the DCB is for XLRI.

To show the record area size in the DD statement, code DCB=LRECL=nK. The value nK may range from 1K to 16383K (expressed in 1024-byte multiples). However, depending on the buffer space available, the value you can specify in most systems will be much smaller than 16383K bytes. This value is used to determine the size of the record area required to contain the largest logical record of the spanned format file.

When using XLRI, the exact LRECL size is communicated in the three low-order bytes of the RDW in the record area. This special RDW format exists only in the record area to communicate the length of the logical record (including the 4-byte RDW) to be written or read. (See the XLRI format of the RDW in Figure 10 on page 26.) DCB LRECL shows the 1024-multiple size of the record area (rounded up to the next nearest kilobyte). The normal DS/DBS SDW format is used at all other times before conversion.

Undefined-Length Records

Format-U permits processing of records that do not conform to the F or V format. Figure 11 shows how each block is treated as a record; therefore, any unblocking that is required 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 do length checking on format-U records, your program may be designed to read less than a complete block into virtual storage.

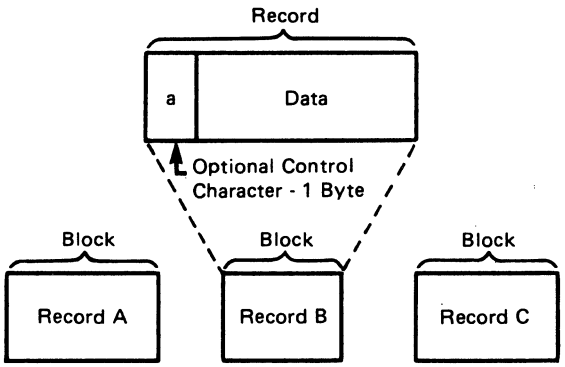


Figure 11. Undefined-Length Records

For format-U records, the user must specify the record length when issuing the WRITE, PUT, or PUTX macro instruction. No length checking is performed by the system, so no error indication will be given if the specified length does not match the buffer size or physical record size.

In update mode, you must issue a GET or READ macro before you issue a PUTX or WRITE macro to a data set on a direct access device. If you change the record length when you issue the PUTX or WRITE macro, the record will be padded with zeros or truncated to match the length of the record received when the GET or READ macro was issued. No error indication will be given.

For Version 3 ISO/ANSI/FIPS tapes, format-U records are not supported. An attempt to process a format-U record from a Version 3 tape will result in entering the label validation installation exit.

ISO/ANSI Version 1 (ISO 1001-1969 and ANSI X3.27-1969) tapes containing format-U records can be used for input only. These records are the same as the format-U records described above, except the control characters must be ISO/ANSI control characters, and block prefixes can be used.

Record Format—Device Type Considerations

Before executing your program, you must supply the operating system with the record format (RECFM) and device-dependent 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 are contained in *JCL*.

The record format (RECFM) parameter of the DCB macro specifies the characteristics of the records in the data set as fixed-length (RECFM=F), variable-length (RECFM=V or D), variable-spanned (RECFM=DS or -VS), or undefined-length (RECFM=U). All record formats except U can be blocked. Fixed-length blocked records (RECFM=FB) can be specified as standard (RECFM=FBS), meaning that 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 are described under "Fixed-Length Records, Standard Format" on page 14.

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" on page 183. The device-dependent (DEVD) parameter of the DCB macro specifies the type of device where the data set's volume resides:

- TA magnetic tape

- PR printer
- PC card punch
- RD card reader
- DA direct access device or Mass Storage System (MSS) virtual volumes

Note: Because the DEVD option is required only for the DCB macro expansion, you are guaranteed the maximum device flexibility by letting it default to DEVD=DA.

Magnetic Tape

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 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 zeros so that the data block length equals 18 bytes. For format-D (ASCII) records, the padding consists of ASCII circumflex characters, which are equivalent to X'5E's.

Note that there is no minimum requirement for block size. However, in nonreturn-to-zero-inverted mode, if a data check occurs on a magnetic tape device, any record shorter than 12 bytes in a read operation will be treated as a noise record and lost. No check for noise will be made unless a data check occurs.

Figure 12 shows how the tape density (DEN) specifies the recording density in bits per inch per track. When DEN is not specified, the highest density capable by the unit will be used.

Recording Density

DEN	7-Track Tape	9-Track Tape	18-Track Tape
1	556 (NRZI)	N/A	N/A
2	800 (NRZI)	800 (NRZI)[1]	N/A
3	N/A	1600 (PE)[2]	N/A
4	N/A	6250 (GCR)[3]	N/A

Notes:

1 NRZI is for nonreturn-to-zero-inverted mode.

2 PE is for phase encoded mode.

3 GCR is for group coded recording mode.

Figure 12. Tape Density (DEN) Values

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

C

Data conversion is to be used. 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.

E

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

T

BCDIC to EBCDIC translation is required.

Card Reader and Punch

Format-F and -U records are acceptable to both the reader and the punch; format-V records are acceptable to the punch only. 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 under "PUT—Write a Record" on page 64. 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 show which bin is to receive the card. If it is not specified, 1 is assumed.

For all QSAM, RECFM=FB, card punch data sets, the block size in the DCB will be adjusted by the system to equal the logical record length. This data set will be treated as RECFM=F. If the system builds the buffers for this data set, the buffer length will be determined by the BUFL parameter. If the BUFL parameter was not specified, the adjusted block size is used for the buffer length.

If the DCB is to be reused with a block size larger than the logical record length, you must reset DCBBLKSI in the DCB and ensure that the buffers are large enough to contain the largest block size expected. You may ensure the buffer size by specifying the BUFL parameter before the first time the data set is opened or by issuing the FREEPOOL macro after each CLOSE macro so the system will build a new buffer pool of the correct size each time the data set is opened.

Punch error correction on the IBM 2540 Card Read Punch is not performed.

The IBM 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. For more information on programming for the 3525 Card Punch, see *OS and OS/VS Programming Support for the IBM 3505 Card Reader and IBM 3525 Card Punch.*

Printer

With the IBM 3800 Printing Subsystem, the data in the record can contain two optional bytes—the optional control character used for carriage control, followed by an optional table reference character used for dynamically selecting a character arrangement table during printing. These characters are discussed below.

Carriage Control Character

You may specify in the DD statement, the DCB macro, 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 show a carriage control function 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. Note that buffer areas must be large enough to accommodate the character. If the immediate destination of the record is a device, such as a 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" on page 183.

3800 Table Reference Character

The 3800 table reference character is a numeric character (0, 1, 2, or 3) corresponding to the order in which the character arrangement table names have been specified with the CHARS keyword. It is used for selection of a character arrangement table during printing. (For more information on the table reference character, see *IBM 3800 Printing Subsystem Programmer's Guide.*)

A numeric table reference character (such as 0) selects from within the table that font to which the character corresponds. The characters' number values represent the order in which the font names have been specified with the CHARS parameter. In addition to using table reference characters to correspond to font names specified on the CHARS parameter, you can also code table reference characters that correspond to font names specified in PAGEDEF control structure. Valid table reference characters vary and range between 0 and 126. Table reference characters with values greater than 126 default to a value of 0 (zero). For additional information, see *IBM 3800 Printing Subsystem Programmer's Guide.*

Record Formats

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. The system does not position the printer to channel 1 for the first record unless specified by a carriage control character.

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 block size should not exceed the length of one line on the printer.

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

For all QSAM, RECFM=FB, printer data sets, the block size in the DCB will be adjusted by the system to equal the logical record length. This data set will be treated as RECFM=F. If the system builds the buffers for this data set, the buffer length will be determined by the BUFL parameter. If the BUFL parameter was not specified, the adjusted block size is used for the buffer length.

If the DCB is to be reused with a block size larger than the logical record length, you must reset DCBBLKSI in the DCB and ensure that the buffers are large enough to contain the largest block size expected. You may ensure the buffer size by specifying the BUFL parameter before the first time the data set is opened or by issuing the FREEPOOL macro after each CLOSE macro so the system will build a new buffer pool of the correct size each time the data set is opened.

Direct Access Device

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 and 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 "Direct Access Volumes" on page 5.

Chapter 4. Selecting an Access Method

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 responsibility is 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 (queued or basic) used to gain access to the data.

Overview of Access Methods

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) and visa versa. If the queued access technique is used to process a sequential data set, the access method is called 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 13.

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

Figure 13. 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 *System-Data Administration*.

Temporary data sets can be handled by a function called virtual I/O (VIO). Data sets for which VIO is specified are located in external page storage. However, to the access methods (BDAM, BPAM, BSAM, QSAM, and EXCP), the data sets appear to reside on a real direct access storage device. VIO provides these advantages:

- Elimination of some of the usual I/O device allocation and data management overhead for temporary data sets
- Generally more efficient use of direct access storage space

To use VIO, you must specify VIO=YES in the UNITNAME macro during system generation, and you must specify a unitname (defined in the UNITNAME macro) on the DD statement for your data set. For additional information on VIO, see *Initialization and Tuning Guide*. For information on the UNITNAME macro, see *System Generation Reference*. For information on changes to the DD statement, see *JCL*.

Basic Direct Access Method (BDAM)

Before you use the BDAM access method to process a data set, consider these implications:

- You create a BDAM data set with the basic sequential access method (BSAM). A special operand in the BSAM DCB macro **(MACRF=WL)** shows that you want to create a BDAM data set.
- The problem program must synchronize all I/O operations with a CHECK or a WAIT macro.
- The problem program must block and unblock its own input and output records. (BDAM only reads and writes data blocks.)
- You can find data blocks within a data set with one of the following addressing techniques:
  - Actual device addresses.
  - Relative track address technique. This locates a track on a direct access device relative to the beginning of the data set.
  - Relative block address technique. This locates a fixed-length data block relative to the beginning of the data set.

For more information about coding the DCB macro to process a BDAM data set, see *Data Administration: Macro Instruction Reference*.

Basic Indexed Sequential Access Method (BISAM)

Before you use the BISAM access method to process an ISAM data set, consider these implications:

- BISAM accesses only ISAM data sets.
- BISAM cannot be used to create an indexed sequential access method (ISAM) data set.
- BISAM directly retrieves logical records by key, updates blocks of records in-place, and inserts new records in their correct key sequence.
- The problem program must synchronize all I/O operations with a CHECK or a WAIT macro.
- Other **DCB** operands are available to reduce input/output operations by defining work areas that contain the highest level master index and the records being processed.

For more information about coding the DCB macro to process a BISAM data set, see *Data Administration: Macro Instruction Reference.*

Basic Partitioned Access Method (BPAM)

BPAM is a subset of BSAM, which processes only the directory of a partitioned data set. BSAM processes the data set members.

Before you use the BPAM access method to process a data set, consider these implications:

- One complete partitioned data set must be on one direct-access volume, but you can concatenate multiple input data sets that are on the same or different volumes.
- When you create a partitioned data set, you must specify the **SPACE** parameter on your first (or only) DD statement for the data set. This parameter defines the size of the data set and its directory so that the system can allocate data set space and pre-format the directory.
- You can use either the basic sequential access method (BSAM) or the queued sequential access method (QSAM) to add or retrieve a BPAM data set member without specifying the BLDL, FIND, or the STOW macro by coding the **DSORG=PS** operand in the DCB macro. (Data set positioning and directory maintenance are then handled by the OPEN and CLOSE macros.) But, be advised that you are really processing the member as if it were part of a sequential data set, so you are not using the complete capabilities of BPAM.
- You can use the STOW macro to add, delete, change, or replace an element name or alias in the directory.
- You can process multiple data set members by passing a list of members to BLDL. Then you can use the FIND macro to position to to a member before processing it.

For more information about coding the DCB macro to process a BPAM data set, see *Data Administration: Macro Instruction Reference.*

Basic Sequential Access Method (BSAM)

Before you use the BSAM access method to process a data set, consider these implications:

- The problem program must block and unblock its own input and output records. (BSAM only reads and writes data blocks.)
- The problem program must manage its own input and and output buffers. It must give BSAM a buffer address with the READ macro, and it must fill its own output buffer before issuing the WRITE macro.
- The problem program must synchronize its own I/O operations by issuing a CHECK macro for each READ and WRITE macro issued.
- BSAM lets you process nonsequential blocks by repositioning with the NOTE and POINT macros.
- You can read and write direct access device record keys with BSAM.

For more information about coding the DCB macro to process a BSAM data set, see *Data Administration: Macro Instruction Reference.*

Queued Indexed Sequential Access Method (QISAM)

Before you use the QISAM access method to process an ISAM data set, consider these implications:

- The characteristics of a QISAM data set are established when the data set is created. You can't change them without reorganizing the data set. The DCB operands that establish these characteristics are: **BLKSIZE, CYLOFL, KEYLEN, LRECL, NTM, OPTCD, RECFM,** and **RKP**.
- A QISAM data set can consist of unblocked fixed-length records (**F**), blocked fixed-length records (**FB**), unblocked variable-length records (**V**), or blocked variable-length records (**VB**).
- QISAM can create an indexed sequential data set (QISAM, load mode), add additional data records at the end of the existing data set (QISAM, resume load mode), update a record in place, or retrieve records sequentially (QISAM, scan mode).
- You can't use track overflow to create or extend an ISAM data set.
- When you create an indexed sequential data set, you can allocate space for the data set's prime area, overflow area, and its cylinder/master index(es) on the same or separate volumes. For more information about space allocation, see *JCL.*

- QISAM automatically generates a track index for each cylinder in the data set and one cylinder index for the entire data set. Specify the DCB operands **NTM** and **OPTCD** to show that the data set requires a master index(es). QISAM creates and maintains as many as three levels of master indexes.

- You can purge records by specifying the **OPTCD=L** DCB option when you create an ISAM data set. This option flags the records you want to purge with a X'FF' in the first data byte of a fixed-length record or the fifth byte of a variable-length record. QISAM ignores these flagged records during sequential retrieval.

- You can get reorganization statistics by specifying the **OPTCD=R** DCB option when an ISAM data set is created. The problem program uses these statistics to determine the status of the the data set's overflow areas.

- When you create an ISAM data set, you must write the records in ascending key order.

For more information about coding the DCB macro to process a QISAM data set, see *Data Administration: Macro Instruction Reference.*

Queued Sequential Access Method (QSAM)

Before you use the QSAM access method to process a data set, consider these implications:

- You can use QSAM to process all record formats except blocks with keys.

- QSAM blocks and unblocks records for you automatically.

- QSAM manages all aspects of I/O buffering for you automatically. The GET macro retrieves the next sequential logical record from the input buffer, and the PUT macro places the next sequential logical record in the output buffer.

- QSAM gives you three transmittal modes: move, locate, and data. These modes give you greater flexibility managing buffers and moving data.

For more information about coding the DCB macro to process a QSAM data set, see *Data Administration: Macro Instruction Reference.*

Chapter 5. Specifying a Data Control Block and Initializing Data Sets

Before processing can begin, you must identify the characteristics of a data set, the volume on which it resides, and its processing requirements. During execution, this 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 change 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 14 on page 40).

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

- Completes the data control block
- Loads all necessary access method routines not already in virtual storage
- Initializes data sets by reading or writing labels and control information
- Builds 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 change most data control block fields either before the data set is opened or when the operating system returns control to your program (at the data control block open exit). Some fields can be changed during processing.

Figure 14 on page 40 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 direct access data set is opened (or a magnetic tape with standard labels is opened for INPUT, RDBACK, or INOUT), any field in the JFCB not completed

by a DD statement is filled in from the data set label (if one exists). When opening a magnetic tape for output, the tape label is assumed not to exist or to apply to the current data set unless you specify DISP=MOD and a volume serial number in the volume parameter of the DD statement. Any field not completed in the DCB is filled in from the JFCB. Fields in the DCB can then be completed or changed by your own DCB exit routine. Then all DCB fields are unconditionally merged into corresponding JFCB fields if your data set is opened for output. This is done by specifying OUTPUT, OUTIN, EXTEND, or OUTINX in the OPEN macro instruction. The DSORG field is not merged unless this field contains zeros in the JFCB. If your data set is opened for input (INPUT, INOUT, RDBACK, or UPDAT is specified in the OPEN macro instruction), the DCB fields are not merged unless the corresponding JFCB fields contain zeros.

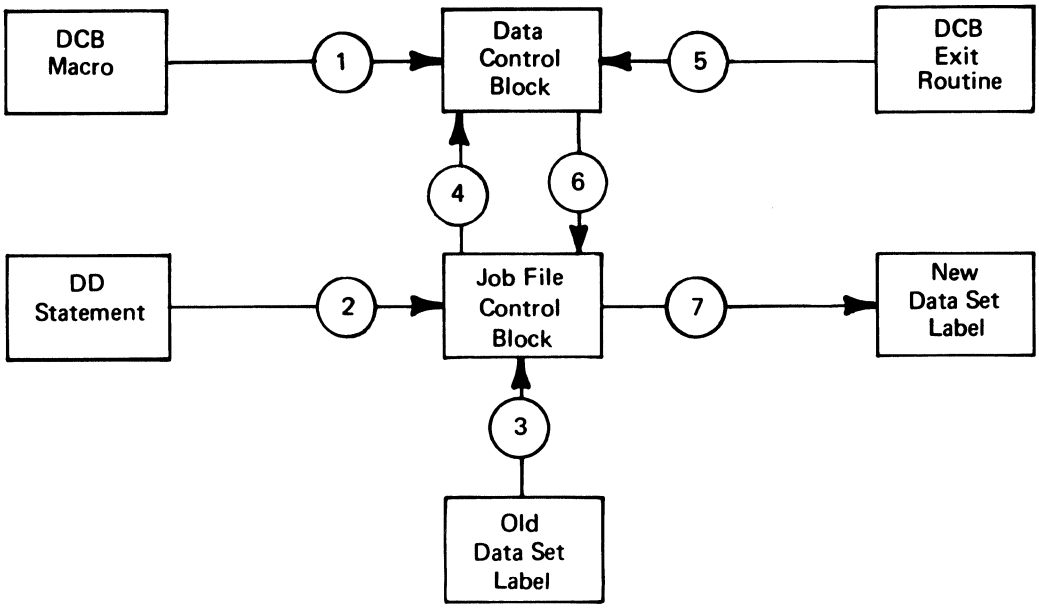


Figure 14. Sources and Sequence of Operations for Completing the Data Control Block

When the data set is closed, the data control block is restored to the condition it had before the data set was opened (the buffer pool is not freed). 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 program terminates, the operating system will close it automatically.

The operating system requires several types of processing information to ensure proper control of your input/output operations. The forms of macros 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 "Managing SAM Buffer Space" on page 89.

Because macros are expanded during the assembly of your program, you must supply the macro forms that are to be used in processing each data set in the associated DCB macro. You can supply buffering requirements and related information in the DCB macro, 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 (EODAD, SYNAD, or user exits) are omitted from the DCB macro, you must complete them in the DCB before they are required.

Note: A data set label to JFCB merge is not performed for concatenated data sets at the end-of-volume time. If you want a merge, turn on the unlike attribute bit (DCBOFPPC) in the DCB. This attribute forces the system through OPEN for each data set in the concatenation, where a label to JFCB merge takes place.

Selecting Data Set Options

For each data set you want 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. Also, the DD statement must supply data set identification, device characteristics, space allocation requests, and related information as specified in *JCL*. 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, or by completing the field yourself before opening the data set.

DCB Parameters

After you have specified the data set characteristics in the DCB macro, you can change them only by changing the DCB during execution. The fields of the DCB discussed below are common to most data organizations and access techniques. (For more information about the DCB fields, see *Data Administration: Macro Instruction Reference*.)

Block Size (BLKSIZE): Specifies the maximum length, in bytes, of a data block. If the records are of format F, the block size must be an integral multiple of the record length, except for SYSOUT data sets. (See Chapter 8, "Spooling and Scheduling Data Sets" on page 75.) If the records are of format V, the block size specified must be the maximum block size. If format-V records are unblocked, the block size must be 4 bytes greater than the record length (LRECL). When spanned variable-length records are specified, the block size is independent of the record length. For ISO/ANSI/FIPS Version 3 records, the maximum block size is 2048.

There is no minimum requirement for block size; however, if a data check occurs on a magnetic tape device, any block 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 maximum block size for an ISO/ANSI/FIPS Version 3 tape is 2048 bytes. This limit may be overridden by a label validation installation exit. (See *Data Facility Product: Customization*.)

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 is processed using 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. 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.

When creating a direct data set, the DSORG in the DCB macro must specify PS or PSU and the DD statement must specify DA or DAU.

Key Length (KEYLEN): Specifies the length (0 to 255) in bytes of an optional key that precedes each block on a direct access device. The value of KEYLEN is not included in BLKSIZE or LRECL but must be included in BUFL if buffer length is specified. Thus, BUFL=KEYLEN+BLKSIZE.

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. For the extended logical record interface for ISO/ANSI/FIPS variable spanned records, LRECL must be specified as LRECL=0K or LRECL=nK.

Record Format (RECFM): Specifies the characteristics of the records in the data set as fixed-length (F), variable-length (V), ISCII/ASCII variable-length (D), or undefined-length (U). Blocked records are specified as FB, VB, or DB. Spanned records are specified as VS, VBS, DS, or DBS. (ISCII/ASCII records are specified as DS or DBS.) 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).

The type of print control can be specified to be in ANSI format-A or in machine code format-M, as described in Appendix B, "Control Characters" on page 183.

Write Validity Check Option (OPTCD=W): You can specify the write validity check option in either the DCB parameter of the DD statement or the DCB macro. 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. Be aware that the write validity check process requires an additional revolution of the device for each record. If the system detects any errors, it starts its standard error recovery procedure.

For buffered tape devices, the write validity check option delays the device end interrupt until the data is physically on tape. When you use the write-validity-check option, you get none of the performance benefits of buffering.

DD Statement Parameters

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. Also, data set identification and disposition, and 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. (See *JCL*.)

Device Affinity (AFF): Requests that specified data sets be allocated to the same device.

Data Control Block (DCB): Provides, through sub parameters, information to be used to complete those fields of the data control block that were not specified in the DCB macro. This parameter cannot be used to change data control block fields that are specified in the user's program.

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 Disposition (DISP): Describes the status (OLD, NEW, KEEP, or DELETE) of a data set and shows what is to be done with it at the end of the job step.

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

Data Set Label (LABEL): Shows the type and contents of the tape label or labels associated with the data set. The operating system verifies standard labels. Standard labels include those specified in the DD statement as SL (standard labels), SUL (standard user labels), AL (American National Standard labels), and AUL (American National Standard user labels). Nonstandard labels (NSL) can be specified only if your installation has incorporated into the operating system routines to write and process nonstandard labels.

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, if requested, when your job is finished.

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

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.

Changing the DCB

You can complete or change the DCB during execution of your program. You can also determine data set characteristics from information supplied by the data set labels. You can make changes or additions before you open a data set, after you close it, during the DCB open exit routine, or while the data set is open. Naturally, you must supply the information before it is needed.

Use the data control block DSECT (DCBD) macro to identify the DCB field names symbolically. If you load a base register with the DCB address, you can refer to any field symbolically.

The DCBD macro generates a dummy control section (DSECT) named IHADCB. Each field name symbol consists of DCB followed by the first 5 letters of the keyword operand for the DCB macro. For example, the symbolic name of the block size operand field is DCBBLKSI. (For other DCB field names, see *Data Administration: Macro Instruction Reference.*)

The attributes of each DCB field are defined in the dummy control section. Use the DCB macro's assembly listing to determine the length attribute and the alignment of each DCB field.

You can code the DCBD macro once to describe all DCBs.

Changing an Address in the Data Control Block: Figure 15 on page 44 shows you how to change a field in the data control block.

```
          ....
                OPEN      (TEXTDCB,INOUT),MODE=31
                ...
EOFEXIT         CLOSE     (TEXTDCB,REREAD),MODE=31,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 15. Changing a Field in the Data Control Block

The data set defined by the data control block TEXTDCB is opened for both input and output. When the problem program no longer needs it for input, 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, the base register for IHADCB. Then it moves the address OUTERROR into the DCBSYNAD field of the DCB. Even though DCBSYNAD is a fullword field and contains important information in the high-order byte, change only the 3 low-order bytes in the field.

All unused address fields in the DCB, except DCBEXLST, are set to 1 when the DCB macro is expanded. Many system routines interpret a value of 1 in an address field as meaning no address was specified, so use it to dynamically reset any field you don't need.

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 initialize by issuing the OPEN macro instruction to open a data set. 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 programs 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
- Record format

This information is used by the system to allocate virtual storage space and load the appropriate routines. These routines, the channel programs and buffer areas created automatically by the system, remain in virtual 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 virtual and auxiliary storage.

Using a Parameter List with 31-bit Addresses

You can code OPEN and CLOSE with **MODE=31** to specify a long form parameter list that can contain 31-bit addresses. To use this long form parameter list, you must be operating in 31-bit addressing mode. The default, **MODE=24**, specifies a standard form parameter list with 24-bit addresses. If **TYPE=J** is specified, you must use the standard form parameter list.

The standard form parameter list is 4 bytes per entry. The standard form parameter list must reside below 16M, but the calling program may be above 16M.

The long form parameter list can reside above or below 16M. Each entry is 8 bytes long. Option information is contained in the first byte, zeros in the next three bytes, and the address of the ACB or DCB is contained in the last four bytes. Although you may code **MODE=31** on the OPEN or CLOSE call for a DCB, the DCB must reside below 16M. All non-VSAM and non-VTAM ACBs must also reside below 16M. Because the address is below 16M, the leading byte of the 4-byte ACB or DCB address must contain zeros. If the byte contains something other than zeros, an error message is issued. If an OPEN was attempted, the data set is not opened. If a CLOSE was attempted, the data set is not closed.

It is up to you to keep the mode specified in the **MF=L** and **MF=E** versions of the OPEN and CLOSE macros consistent. If **MODE=31** is specified in the **MF=L** version of the OPEN or CLOSE macro, **MODE=31** must also be coded in the corresponding **MF=E** version of the macro. Unpredictable results occur if the mode specified is not consistent.

Managing Buffer Pools When Closing Data Sets

After closing the data set, you should issue a FREEPOOL macro instruction to release the virtual 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 "Buffer Pool Construction" on page 89.)

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 ends the task, and data results can be unpredictable. Note, however, that the operating system cannot automatically close any open data sets after the normal end of a program that was brought into virtual storage by the loader. Therefore, loaded programs must include CLOSE macro instructions for all open data sets.

Simultaneous Opening and Closing of Multiple Data Sets

An OPEN or CLOSE macro instruction can be used to begin or end 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. The coding examples in Figure 16 on page 50 and Figure 18 on page 52 show the macro expansions for simultaneous open and close operations.

Opening and Closing Data Sets Shared by More Than One Task

When more than one task is sharing a data set, the following restrictions must be recognized. Failure to adhere to these restrictions endangers the integrity of the shared data set.

- All tasks sharing a DCB must be in the job step that opened the DCB (see "Sharing Data Sets" on page 67).
- Each task sharing a DCB must ensure that all the input and output operations it starts using a given DCB are complete, before the task terminates. A CLOSE macro instruction issued for the DCB will ensure termination of all input and output operations.
- A DCB can be closed only by the task that opened it.

Considerations for Allocating Direct Access Data Sets

When you allocate space for a new data set on a direct access volume, the tracks contain unknown data. A program that tries to access data on these tracks before known data is written on them may get unpredictable results, such as program checks or I/O errors. The program may even appear to run correctly!

If you must access a newly allocated data set before you put known data into it, use one of the following methods to make it appear empty:

1. At allocation time, specify a primary allocation value of zero; such as SPACE=(TRK,(0,10)) or SPACE=(CYL,(0,50)). This method prevents processing certain labels if user labels are requested (LABEL=(,SUL)).
2. After allocation time, run a program that opens the data set for output and closes it without writing anything. This puts an end-of-file mark at the beginning of the data set.

- Two or more DCBs should never be concurrently open for output to the same data set, except with the basic indexed sequential access method (BISAM).
- If, concurrently, one DCB is open for input or update, and one for output to the same data set on a direct access device, the input or update DCB may be unable to read what the output DCB wrote if the output DCB extended the data set.
- 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 other DCB using the same DD statement. This is true for both input and output and is especially important when you are using more than one access method. Any action on one DCB that alters the JFCB affects the other DCBs and thus can cause unpredictable results. 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 IBM 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, after a data set has been closed, I/O operations cannot be performed on any of the associated data sets. See *Programming Support for the IBM 3505 Card Reader and the IBM 3525 Card Punch* for more information.
- The OPEN macro gets user control blocks and user storage in the protection key that is specified in the TCB(TCBPKF). Therefore, any task that processes the DCB (such as Open, Close, or EOV) must be in the same protection key specified in the TCB, or must be in key 0. Also, the Open and Close must be done in the same key.

  Note: If the Open is done while processing in key 0, then user storage obtained by Open will be from subpool 252.
- 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 on the availability of devices at a particular time. Additional information on volume disposition is provided in *JCL*.

Open/Close/EOV Errors

There are two classes of errors that can occur during open, close, and end-of-volume processing: determinate and indeterminate errors. Determinate errors are errors associated with a system completion code. For example, a condition associated with the 213 completion code with a return code of 04 might be detected during open processing, indicating that a format-1 DSCB could not be found for a data set being opened. Indeterminate errors are errors that cannot be anticipated, such as program checks.

If a determinate error occurs during the processing resulting from a concurrent OPEN or CLOSE macro instruction, an attempt will be made to complete open or close processing of the DCBs that are not associated with the DCB in error. Note

that you can also choose to abnormally end the task immediately by coding a DCB abend exit routine that shows the "immediate termination" option (see -- Heading id 'abend' unknown --). When all open or close processing is completed, abnormal end processing is begun. Abnormal end involves forcing all DCBs associated with a given OPEN or CLOSE macro to close status, thereby freeing all storage devices and other system resources related to the DCBs.

If an indeterminate error (such as a program check) occurs during open, close, or EOV processing, no attempt is made by the system control program to complete concurrent open or close processing. The DCBs associated with the OPEN or CLOSE macro are forced to close status if possible, and the resources related to each DCB are freed.

To determine the status of any DCB after an error, the OPEN (CLOSE) return code in register 15 must be interrogated for the following values:

Return Code	Meaning
00 (X'00')	All entries in the parameter list opened successfully.
04 (X'04')	All entries in the parameter list have successfully completed open, but one or more entries have a warning message.
08 (X'08')	One or more entries in the parameter list were not opened successfully. The entries with errors are restored to their preopen status.
12 (X'0C')	One or more entries in the parameter list were not opened successfully.

The entries with errors are not restored, and cannot be reopened without restoration.

During task termination, the system issues a CLOSE macro for each data set that is still open. If this is an abnormal termination for QSAM, the close routines that would normally finish processing buffers are bypassed. Any outstanding I/O requests are purged. Thus, your last data records may be lost for a QSAM output data set.

It is a good procedure to close an ISAM data set before task termination because, if an I/O error is detected, the ISAM close routines cannot return the problem program registers to the SYNAD routine, causing unpredictable results.

Installation exits

Four installation exit routines are provided for abnormal end with ISO/ANSI/FIPS Version 3 tapes.

- The label validation exit is entered during open/EOV if an invalid label condition is detected, and label validation has not been suppressed. Invalid conditions include incorrect alphameric fields, nonstandard values (for example, RECFM=U, block size greater than 2048, or a zero generation number), invalid label sequence, nonsymmetrical labels, invalid expiration date sequence, and duplicate data set names.

- The validation suppression exit is entered during open/EOV if volume security checking has been suppressed, if the volume label accessibility field contains an

ISCII/ASCII space character, or if RACF accepts a volume and the accessibility field does not contain an uppercase A through Z.

- The volume access exit is entered during open/EOV if a volume is not RACF protected and the accessibility field in the volume label contains an ISCII/ASCII uppercase A through Z.

- The file access exit is entered after positioning to a requested data set if the accessibility field in the HDR1 label contains an ISCII/ASCII uppercase A through Z.

For additional information about ISO/ANSI/FIPS Version 3 installation exits, see *Data Facility Product: Customization.*

OPEN—Prepare a Data Set for Processing

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

Processing Method

You can process a data set as either input or output. This is done by coding INPUT, OUTPUT, or EXTEND as the processing method operand of the OPEN macro. For BSAM, code INOUT, OUTIN, or OUTINX. If the data set resides on a direct access volume, you can code UPDAT in the processing method operand to show that records can be updated. By coding RDBACK in this operand, you can specify that a magnetic tape volume containing format-F or format-U records is to be read backward. (Variable-length records cannot be read backward.) 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 or EXTEND when you are using the queued indexed sequential access method (QISAM) to create an indexed sequential data set. You can override the INOUT, OUTIN, UPDAT, or OUTINX at execution time by using the LABEL parameter of the DD statement, as discussed in *JCL.*

Note: Unless label validation has been suppressed, OPEN for MOD (OLD OUTPUT/OUTIN), INOUT, EXTEND, or OUTINX cannot be processed for ISO/ANSI/FIPS Version 3 tapes, because this kind of processing updates only the closing label of the file, causing a label symmetry conflict. An unmatching label should not frame the other end of the file.

SYSIN and SYSOUT data sets must be opened for INPUT and OUTPUT, respectively. INOUT is treated as INPUT; OUTIN, EXTEND, or OUTINX is treated as OUTPUT. UPDAT and RDBACK cannot be used.

In Figure 16 on page 50, the data sets associated with three DCBs are to be opened simultaneously.

```
        OPEN    (TEXTDCB,,CONVDCB,(OUTPUT),PRINTDCB,
                (OUTPUT))
+       CNOP    0,4                 Align list to fullword
+       BAL     1,*+16              Load reg1 w/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 16. Opening Three Data Sets Simultaneously

Because 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 disposition indicated by the DD statement DISP parameter is used.

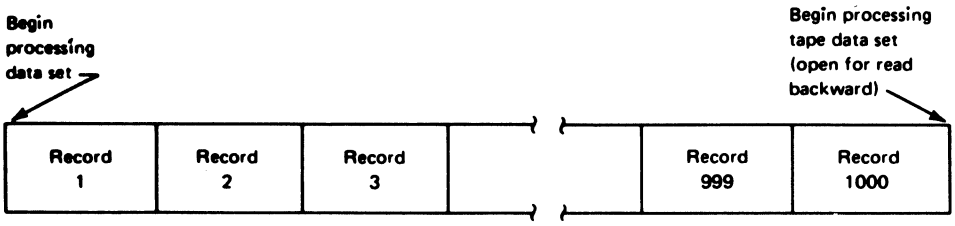
At execution, the SVC 19 instruction passes control to the open routine, which initializes the three DCBs and loads the appropriate access method routines.

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 (tapes only) 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.

You can code CLOSE TYPE=T and perform some close functions for sequential data sets on magnetic tape and direct access volumes processed with BSAM. When you use TYPE=T, the DCB used to process the data set maintains its open status. You don't have to issue another OPEN macro instruction to continue processing the same data set. This option cannot be used in a SYNAD routine.

The TYPE=T operand causes the system control program to process labels, modify some of the fields in the system control blocks for that data set, and reposition the volume (or current volume in the case of multivolume data sets) in much the same way that the normal CLOSE macro does. When you code TYPE=T, you can specify that the volume is either to be positioned at the end of data (the LEAVE option) or to be repositioned at the beginning of data (the REREAD option). Magnetic tape volumes are repositioned either immediately before the first data record or immediately after the last data record; the presence of tape labels has no effect on repositioning. Figure 17 on page 51, which assumes a sample data set containing 1000 records, illustrates the relationship between each positioning option and the point where you resume processing the data set after issuing the temporary close.



If you CLOSE TYPE = T and specify	After temporary close, you will resume processing
LEAVE	Immediately after record 1000
LEAVE (with tape data set open for read backward)	Immediately before record 1
REREAD	Immediately before record 1
REREAD (with tape data set open for read backward)	Immediately after record 1000

Figure 17. Record Processed When LEAVE or REREAD Is Specified for CLOSE TYPE=T

If you code the release (RLSE) operand on the DD statement for an output data set, it is ignored by temporary close (CLOSE TYPE=T). However, if the last operation was a write, then normal close (without TYPE=T) releases any unused space.

Space is released on a track boundary if the extent containing the last record was allocated in units of tracks or in units of average block lengths with ROUND not specified. Space is released on a cylinder boundary if the extent containing the last record was allocated in units of cylinders or in units of average block lengths with ROUND specified. However, a cylinder boundary extent may be released on a track boundary if:

- The DD statement used to access the data set contains a space parameter specifying units of tracks or units of average block lengths with ROUND not specified, or
- No space parameter is supplied in the DD statement and no secondary space value has been saved in the format-1 DSCB for the data set. (This may occur if the release indicator is set in the JFCB through RDJFCB and OPEN TYPE=J, or through the OPEN installation exit.) In this case, the performance benefit of cylinder boundaries is lost.

For data sets processed with BSAM, you can use CLOSE TYPE=T with the following restrictions:

- The DCB for the data set you are processing on a direct access device must specify either DSORG=PS or DSORG=PSU for input processing, and either DSORG=PS, DSORG=PSU, DSORG=PO, or DSORG=POU for output processing.

- The DCB must not be open for input to a member of a partitioned data set.
- If you open a data set on a direct access device for output and issue CLOSE TYPE=T, the volume will be repositioned only if the data set was created with DSORG=PS, DSORG=PSU, DSORG=PO, or DSORG=POU (you cannot specify the REREAD option if DSORG=PO or DSORG=POU is specified). (This restriction prohibits the use of temporary close following or during the building of a BDAM data set that is created by specifying BSAM MACRF=WL.)
- If you open the data set for input and issue CLOSE TYPE=T with the LEAVE option, the volume will be repositioned only if the data set specifies DSORG=PS or DSORG=PO.

Note: When a data control block is shared among multiple tasks, only the task that opened the data set can close it unless TYPE=T is specified.

Before issuing the CLOSE macro, a CHECK macro must be issued for all DECBs that have outstanding I/O from WRITE macro instructions. When CLOSE TYPE=T is specified, a CHECK macro must be issued for all DECBs that have outstanding I/O from either WRITE or READ macro instructions.

In Figure 18, the data sets associated with three DCBs are to be closed simultaneously.

```
         CLOSE    (TEXTDCB,,CONVDCB,,PRINTDCB)
+        CNOP     0,4                 Align list to fullword
+        BAL      1,*+16              Load regl w/list addr
+        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 18. Closing Three Data Sets Simultaneously

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

Volume Positioning

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

Releasing Data Sets and Volumes

You are offered the option of being able to release data sets and the volumes the data sets reside on when your task is no longer using them. If you are not sharing data sets, these data sets would otherwise remain unavailable for use by other tasks until the job step that opened them is terminated.

There are two ways to code the CLOSE macro instruction that can result in releasing a data set and the volume on which it resides at the time the data set is closed:

In conjunction with the FREE=CLOSE parameter of the DD statement, you can code:

```
CLOSE    (DCB1,DISP) or
CLOSE    (DCB1,REWIND)
```

If you do not code FREE=CLOSE on the DD statement, you can code:

```
CLOSE    (DCB1,FREE)
```

See *JCL* for information about how to use and code the FREE=CLOSE parameter of the DD statement.

In either case, tape data sets and volumes are freed for use by another job step. Data sets on direct access devices will be freed and the volumes on which they reside will be freed if no other data sets on the volume are open. Additional information on volume disposition is provided in *JCL*.

Data sets being temporarily closed (using CLOSE TYPE=T) cannot be released at the time the data set is closed. They will be released at the end of the job step.

For additional information and coding restrictions on the CLOSE macro, see *Data Administration: Macro Instruction Reference*.

End-of-Volume Processing

The access methods pass control to the data management end-of-volume routine when any of the following conditions is detected:

- Tapemark (input tape volume).
- Filemark or end of last extent (input direct access volume).
- End-of-data indicator (input device other than magnetic tape or direct access volume). An example of this would be the last card read on a card reader.
- 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.

If the LABEL parameter of the associated DD statement shows standard labels, the end-of-volume routine checks or creates standard trailer labels. If SUL or AUL is specified, control is 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 for Tapes

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 magnetic tape can be specified as LEAVE or REREAD.

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.

If, however, you want to position the current volume according to the option specified in the DISP parameter of the DD statement, you code DISP in the OPEN macro instruction.

DISP
specifies that a tape volume is to be disposed of in the manner implied by the DD statement associated with the data set. Direct access volume positioning and disposition are not affected by this parameter of the OPEN macro instruction. There are several dispositions that can be specified in the DISP parameter of the DD statement; DISP can be PASS, DELETE, KEEP, CATLG, or UNCATLG.

The resultant action at the time an end-of-volume condition arises depends on (1) how many tape units are allocated to the data set and (2) how many volumes are specified for the data set in the DD statement. This is determined by the UNIT and VOLUME parameters of the DD statement associated with the data set. If the number of volumes is greater than the number of units allocated, the current volume will be rewound and unloaded. If the number of volumes is less than or equal to the number of units, the current volume is merely rewound.

For magnetic tape volumes that are not being unloaded, 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 start the 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.

If an FEOV macro is issued for a spanned multivolume data set that is being read using QSAM, errors may occur when the next GET macro is issued. These errors are documented in "Spanned Format-VS Records (Sequential Access Method)" on page 19.

The FEOV macro instruction can only be used when you are using BSAM or QSAM. FEOV is ignored if issued for a SYSIN or SYSOUT data set.

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 a 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 configuration and processing on a different configuration. For example, an IBM 3330 Disk Storage drive can be used as a substitute for several magnetic tape units.

Device independence is not difficult to achieve, but requires some programming considerations. See below.

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 in which 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, a printer, or a spooled data set.

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

- Omit all device-dependent macros 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 or during the open exit routine.

In examining the following list of macros, 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 as a reference to magnetic tape, that is, you must create a new data set rather than attempt to update.

OPEN
: Specify INPUT, OUTPUT, INOUT, OUTIN, OUTINX, or EXTEND. The parameters RDBACK and UPDAT 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 or modify existing 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.

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. Also, you must consider whether you expect to process XLRI records.

DSORG
Specify sequential organization (PS or PSU) to get the full DCB expansion.

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.

Chapter 6. Accessing Records in Data Sets

Accessing Data with READ/WRITE

The basic access technique provides the READ and WRITE macro instructions for transmitting data between virtual and auxiliary storage. This technique is used when you want to process records other than sequentially 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 unblocking of records are 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 start 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). (The only exception to this is that, when the SYNAD or EODAD routine is entered, a CHECK macro instruction should not be issued for outstanding READ or WRITE requests.) 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).

Grouping Related Control Blocks in a Paging Environment

Related control blocks (the DCB and DECB) and data areas (buffers and key areas) should be coded so they assemble in the same area of your program. This will reduce the number of paging operations required to read from and write to your data set.

Note: DCB, DECB, and buffers must reside below 16 megabytes.

Using Overlapped I/O with BSAM

When using BSAM with overlapped I/O (multiple I/O requests outstanding at one time), more than one DECB must be used. A different DECB should be specified for each channel program. For example, if you specify NCP=3 in your DCB for the data set and you are reading records from the data set, you should code the following macros in your program:

```
....
.READ DECB1,...
.READ DECB2,...
.READ DECB3,...
.CHECK DECB1
.CHECK DECB2
.CHECK DECB3
....
```

READ—Read a Block

The READ macro retrieves a data block from an input data set and places it in a designated area of virtual 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 virtual storage and the address of the record is returned to you in the DECB.

When you use the READ macro 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, preceded in the buffer by its key, or a subsequent segment that does not have a key. This procedure is called offset reading.

You can specify variations of the READ macro 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 with the last block truncated 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 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 places a data block in an output data set from a designated area of virtual storage. The WRITE macro 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 ISCII/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 area to be translated from EBCDIC to ISCII/ASCII.

As with the READ macro, you can specify variations of the WRITE macro 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.

Indexed Sequential

K Write a block containing an updated record, or replace a record with a fixed, unblocked record having the same key. The record to be replaced need not have been read into virtual storage.

KN Write a new record or change the length of a variable-length record.

Direct

SD Write a dummy fixed-length record. (BDAM load mode)

SZ Write a capacity record (R0). The system supplies the data, writes the capacity record, and advances to the next track. (BDAM load mode)

SFR Write the data set sequentially with next-address feedback. (BDAM load mode variable spanned)

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. The system tests for errors and exceptional conditions in the data event control block (DECB). Successive CHECK macros issued for the same data set must be issued in the same order as the associated READ and WRITE macros.

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 starts the 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 ISCII/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. 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.

For BDAM and BISAM, a WAIT macro must be issued for each READ or WRITE macro if MACRF=C is not coded in the associated DCB. When MACRF=C is coded, and always for BSAM and BPAM, a CHECK macro must be issued for each READ or WRITE macro. Because the CHECK macro incorporates the function of the WAIT macro, a WAIT is normally redundant for those access methods. The ECBLIST form of the WAIT macro may be useful, though, in selecting which of several outstanding events should be checked first.

The WAIT macro can be used to await completion of multiple read and write operations. Each operation must then be checked or tested separately. **Example:** You have opened an input DCB for BSAM with NCP=2, and an output DCB for BISAM with NCP=1 and without specifying MACRF=C. You have issued two BSAM READ macros and one BISAM WRITE macro. You now issue the WAIT macro with ECBLIST pointing to the BISAM DECB and the first BSAM DECB. (Because BSAM requests are serialized, the first request must execute before the second.) When you regain control, you will inspect the DECBs to see which has completed (second bit on). If it was BISAM, you will issue another WRITE macro. If it was BSAM, you will issue a CHECK macro and then another READ macro.

Data Event Control Block (DECB)

A data event control block is a 16- to 32-byte area reserved by each READ or WRITE macro. It contains the ECB, control information, and pointers to control blocks. The DCB is described in Appendix A of *Data Administration: Macro Instruction Reference*.

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.

Accessing Data with GET/PUT

The queued access technique provides GET and PUT macros for transmitting data within virtual storage. These macro instructions cause automatic blocking and unblocking of the records stored and retrieved. Anticipatory (look-ahead) buffering and synchronization (overlap) of input and output operations with instruction stream processing are automatic features of the queued access method.

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 storage at a 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 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 is used to obtain a record from an input data set. It operates in a logical sequential and device-independent manner. As required, the GET macro schedules the filling of input buffers, unblocks 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 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 ISCII/ASCII to EBCDIC.

PUT—Write a Record

The PUT macro is used to write a record into an output data set. Like the GET macro, it operates in a logical sequential and device-independent manner. As required, the PUT macro blocks records, schedules the emptying of output buffers, and handles output error correction procedures. For sequential data sets, it also starts 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 ISCII/ASCII.

If the PUT macro 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 block size (BLKSIZE) to provide an optimum block size if the records are temporarily placed on magnetic tape or a direct access volume.

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

PUTX—Write an Updated Record

The PUTX macro 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.

When you use the PUTX macro to update, each record is returned to the data set referred to by a previous locate mode GET macro instruction. The buffer containing the updated record is flagged and written back to the same location on the direct access storage device where 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.

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

Parallel Input Processing (QSAM Only)

QSAM parallel input processing may be used to process two or more input data sets concurrently, such as sorting or merging several data sets at the same time. This eliminates the need for issuing a separate GET macro to each DCB processed. The get routine for parallel input processing selects a DCB with a ready record and then transfers control to the normal get routine. If there is no DCB with a ready record, a multiple WAIT macro is issued.

Parallel input processing provides a logical input record from a queue of data sets with equal priority. The function supports QSAM with input processing, simple buffering, locate or move mode, and fixed-, variable-, or undefined-length records. Spanned records, track-overflow records, dummy data sets, and SYSIN data sets are not supported.

Parallel input processing can be interrupted at any time to retrieve records from a specific data set, or to issue control instructions to a specific data set. When the retrieval process has been completed, parallel input processing may be resumed.

Data sets can be added to or deleted from the data set queue at any time. It is important to note, however, that, as each data set reaches an end-of-data condition, the data set must be removed from the queue with the CLOSE macro before a subsequent GET macro is issued for the queue; otherwise, the task may be ended abnormally.

A request for parallel input processing is indicated by including the address of a parallel data access block (PDAB) in the DCB exit list. For additional information on the DCB exit list, see Chapter 7, "DCB Exit Routines" on page 73.

With the use of the PDAB macro, you can create and format a work area that identifies the maximum number of DCBs that can be processed at any one time. If you exceed the maximum number of entries indicated in the PDAB macro when adding a DCB to the queue with the OPEN macro, the data set will not be available for parallel input processing; however, it may be available for sequential processing.

When issuing a parallel GET macro, register 1 must always point to a PDAB. You may load the register or let the GET macro do it for you. When control is returned to you, register 1 contains the address of a logical record from one of the data sets in the queue; registers 2 through 13 contain their original contents at the time the GET macro was issued; registers 14, 15, and 0 are changed.

Through the PDAB, you can find the data set from which the record was retrieved. A fullword address in the PDAB (PDADCBEP) points to the address of the DCB. It should be noted that this pointer may be invalid from the time a CLOSE macro is issued to the issuing of the next parallel GET macro.

In Figure 19 on page 66, not more than three data sets (MAXDCB=3 in the PDAB operand) will be open for parallel processing at a time. If data definition statements and data sets are supplied, DATASET1, DATASET2, and DATASET3 will be opened for parallel input processing as specified in the input processing OPEN macro. Other attributes of each data set are QSAM (MACRF=G), simple buffering by default, locate or move mode (MACRF=L or M), fixed-length records (RECFM=F), and exit list entry for a PDAB (X'92'). Note that both locate and move modes may be used in the same data set queue. The mapping macros, DCBD and PDABD, are used to reference the DCBs and the PDAB respectively.

```
            ...
            OPEN   (DATASET1,(INPUT),DATASET2,(INPUT),DATASET3,          X
                   (INPUT),DATASET4,(OUTPUT))
            TM     DATASET1+DCBQSWS-IHADCB,DCBPOPEN  Opened for
                                                     parallel processing
            BZ     SEQRTN                            Branch on no to
                                                     sequential routine
            TM     DATASET2+DCBQSWS-IHADCB,DCBPOPEN
            BZ     SEQRTN
            TM     DATASET3+DCBQSWS-IHADCB,DCBPOPEN
            BZ     SEQRTN
GETRTN      GET    DCBQUEUE,BUFFERAD,TYPE=P
            LR     10,1                              Save record pointer
            ...
            ...                                      Record updated in place
            ...
            PUT    DATASET4,(10)
            B      GETRTN
EODRTN      EQU    *                                 Close DCB which just
                                                     reached EODAD
            L      2,DCBQUEUE+PDADCBEP-IHAPDAB
            CLOSE   ((2))
            CLC    ZEROS(2),DCBQUEUE+PDANODCB-IHAPDAB  Any DCBs left?
            BL     GETRTN                            Branch if yes
            ...
            ...
DATASET1    DCB    DDNAME=DDNAME1,DSORG=PS,MACRF=GL,RECFM=FB,            X
                   LRECL=80,EODAD=EODRTN,EXLST=SET3XLST
DATASET2    DCB    DDNAME=DDNAME2,DSORG=PS,MACRF=GL,RECFM=FB,            X
                   LRECL=80,EODAD=EODRTN,EXLST=SET3XLST
DATASET3    DCB    DDNAME=DDNAME3,DSORG=PS,MACRF=GMC,RECFM=FB,           X
                   LRECL=80,EODAD=EODRTN,EXLST=SET3XLST
DATASET4    DCB    DDNAME=DDNAME4,DSORG=PS,MACRF=PM,RECFM=FB,            X
                   LRECL=80
DCBQUEUE    PDAB   MAXDCB=3
SET3XLST    DC     0F'0',X'92',AL3(DCBQUEUE)
ZEROS       DC     X'0000'
            DCBD   DSORG=QS
            PDABD
            ...
```

Note: The number of bytes required for PDAB is equal to 24 + 8n, where n is the value of the keyword, MAXDCB.

Figure 19. Parallel Processing of Three Data Sets

Following the OPEN macro, tests are made to determine whether the DCBs were opened for parallel processing. If not, the sequential processing routine is given control.

When one or more data sets are opened for parallel processing, the get routine retrieves a record, saves the pointer in register 10, processes the record, and writes it to DATASET4. This process continues until an end-of-data condition is detected on one of the input data sets; the end-of-data routine locates the completed input data set and removes it from the queue with the CLOSE macro. A test is then made to determine whether any data sets remain on the queue. Processing continues in this manner until the queue is empty.

Sharing Data Sets

There are two conditions under which a data set on a direct access device can be shared by two or more tasks:

- Two or more DCBs are opened and used concurrently by the tasks to refer to the same, shared data set (multiple DCBs).
- Only one DCB is opened and used concurrently by multiple tasks in a single job step (a single, shared DCB).

Job control language (JCL) statements and macros are provided in the operating system that help you ensure the integrity of the data sets you want to share among the tasks that process them. Figure 20 on page 68 and Figure 21 on page 70 show which JCL and macros you should use, depending on the access method your task is using and mode of access (input, output, or update).

Figure 20 describes the macros, JCL, and processing procedures you should use if more than one DCB has been opened to the shared data set. The DCBs can be used by tasks in the same or different job steps.

MULTIPLE DCBs

ACCESS MODE	ACCESS METHOD				
	BSAM,BPAM	QSAM	BDAM	QISAM	BISAM
Input	DISP = SHR	DISP = SHR	DISP = SHR	DISP = SHR	DISP = SHR
Output	No facility	No facility	DISP = SHR	No facility	DISP = SHR and ENQ on Data Set
Update	DISP = SHR user must ENQ on block	DISP = SHR and Guarantee discrete blocks	DISP = SHR BDAM will ENQ on block	DISP = SHR and ENQ on data set and guarantee discrete blocks	DISP = SHR and ENQ on data set and guarantee discrete blocks

DISP=SHR:

Each job step sharing an existing data set must code SHR as the subparameter of the DISP parameter on the DD statement for the shared data set to allow the steps to execute concurrently. For additional information about ensuring data set integrity, see *JCL*. If the tasks are in the same job step, DISP=SHR is not required.

No facility:

There are no facilities in the operating system for sharing a data set under these conditions.

ENQ on data set:

Besides coding DISP=SHR on the DD statement for the data set that is to be shared, each task must issue ENQ and DEQ macros naming the data set or block as the resource for which exclusive control is required. The ENQ must be issued before the GET (READ); the DEQ macro should be issued after the PUTX or CHECK macro that ends the operation. (For additional information on the use of ENQ and DEQ macros, see *Supervisor Services and Macro Instructions.*)

Guarantee discrete blocks:

When you are using the access methods that provide blocking and unblocking of records (QSAM, QISAM, and BISAM), it is necessary that every task updating the data set ensure that it is not updating a block that contains a record being updated by any other task. There are no facilities in the operating system for ensuring that discrete blocks are being processed by different tasks.

ENQ on block:

If you are updating a shared data set (specified by coding DISP=SHR on the DD statement) using BSAM or BPAM, your task and all other tasks must serialize processing of each block of records by issuing an ENQ macro before the READ macro and a DEQ macro after the CHECK macro that follows the WRITE macro you issued to update the record. If you are using BDAM, it provides for enqueuing on a block using the READ exclusive option that is requested by coding MACRF=X in the DCB and an X in the type operand of the READ and WRITE macros. (For an example of the use of the BDAM macros, see "Exclusive Control for Updating" on page 123.)

Figure 20. JCL, Macro Instructions, and Procedures Required to Share a Data Set Using Multiple DCBs

Figure 21 on page 70 describes the macros you can use to serialize processing of a shared data set when a single DCB is being shared by several tasks in a job step. The DISP=SHR specification on the DD statement is not required.

Data sets can also be shared both ways at the same time. More than one DCB can be opened for a shared data set, while more than one task can be sharing one of the DCBs. Under this condition, the serialization techniques specified for indexed sequential and direct data sets in Figure 20 satisfy the requirement. For sequential and partitioned data sets, the techniques specified in Figure 20 and Figure 21 must be used.

More information on opening and closing data sets by more than one task is in "Opening and Closing a Data Set" on page 44.

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. Careful planning should be exercised in accessing a shared data set or the same storage area on shared devices by multiple independent systems. Without proper intersystem communication, data integrity could be endangered. For details, see *System Macros and Facilities*.

A SINGLE SHARED DCB

ACCESS MODE	ACCESS METHOD				
	BSAM, BPAM, BDAM Create	QSAM	BDAM	QISAM	BISAM
Input	ENQ	ENQ	No action required	ENQ	ENQ
Output	ENQ	ENQ	No action required	ENQ and key sequence	ENQ
Update	ENQ	ENQ	No action	ENQ	ENQ

ENQ:

When a data set is being shared by two or more tasks in the same job step (all that use the same DCB), each task processing the data set must issue an ENQ macro instruction on a predefined resource name before issuing the macro or macros that begin the input/output operation. Each task must also release exclusive control by issuing the DEQ macro at the next sequential instruction following the input/output macro. If, however, you are processing an indexed sequential data set sequentially using the SETL and ESETL macros, you must issue the ENQ macro before the SETL macro and the DEQ macro after the ESETL macro. Note also that if two tasks are writing different members of a partitioned data set, each task should issue the ENQ macro before the FIND macro and issue the DEQ macro after the STOW macro that completes processing of the member. Additional reference information on the ENQ and DEQ macros is presented in *Supervisor Services and Macro Instructions*. For an example of the use of ENQ and DEQ macros with BISAM, see Figure 43 on page 122.

Figure 21 (Part 1 of 2). Macro Instructions and Procedures Required to Share a Data Set Using a Single DCB

No action required:
See "Sharing Direct Data Sets" on page 127.

ENQ on block:
When updating a shared BDAM data set, every task must use the BDAM exclusive control option that is requested by coding MACRF=X in the DCB macro and an X in the type operand of the READ and WRITE macro instructions. See "Exclusive Control for Updating" on page 123 for an example of the use of BDAM macros. Note that all tasks sharing a data set must share subpool 0 (see the ATTACH macro description in *Supervisor Services and Macro Instructions*).

Key sequence:
Tasks sharing a QISAM load mode DCB must ensure that the records to be written are presented in ascending key sequence; otherwise, a sequence check will result in (1) control being passed to the SYNAD routine identified by the DCB, or (2) if there is no SYNAD routine, termination of the task.

Figure 21 (Part 2 of 2). Macro Instructions and Procedures Required to Share a Data Set Using a Single DCB

Analyzing I/O Errors

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 or in error analysis routines.

SYNADAF—Perform SYNAD Analysis Function

The SYNADAF macro 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 either 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:

```
,TESTJOBb,STEP2bbb,283,TA,MASTERbb,READb,DATA CHECKbbbbb,
0000015,BSAM
```

Note: In the above example, a b indicates a blank.

This message shows that a data check occurred during reading of the 15th 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 to 13) contain binary information. If no data was transmitted or if the access method is QISAM, the first 6 bytes are blanks or binary zeros. 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 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 releases the message and save areas provided by the SYNADAF macro. You must issue this macro instruction before returning from the error analysis routine.

ATLAS—Perform Alternate Track Location Assignment

The ATLAS macro lets your program 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.

The use of this macro requires a knowledge of channel programming. A detailed description of the macro instruction and its use is included in *System-Data Administration*.

If you do not use the ATLAS macro, you can use the IEHATLAS utility program or the Device Support Facilities program to do the same function. The principal difference between ATLAS and the IEHATLAS utility program is that the latter provides error recovery only after your own program has been completed. (For a detailed description of IEHATLAS, see *Utilities*.) The INSPECT command for the Device Support Facilities program can assign alternate tracks. (See *Device Support Facilities User's Guide and Reference*.)

Chapter 7. DCB Exit Routines

The DCB macro 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 or you can complete the DCB fields before opening the data set. Figure 22 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.

Because OPEN/CLOSE/EOV enqueues on SYSZTIOT, functions that require SYSZTIOT cannot be executed in the OPEN/CLOSE/EOV exit routines. Some of these functions are LOCATE, OBTAIN, SCRATCH, and CATALOG.

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
Allocation retrieval list	When issuing a RDJFCB macro instruction	EXLST operand and exit list
Block count	After unequal block count comparison by end-of-volume routine	EXLST operand and exit list
DCB abend	When an abend condition occurs in OPEN, CLOSE, or EOV routine	EXLST operand and exit list
DCB open	When opening a data set	EXLST operand and exit list
Defer input trailer label	When end-of-data is reached	EXLST operand and exit list
Defer nonstandard input trailer label	When end-of-data is reached	EXLST operand and exit list
End-of-volume	When changing volumes	EXLST operand and exit list

Figure 22 (Part 1 of 2). Data Management Exit Routines

Exit Routine	When Available	Where Specified
FCB image	When opening a data set or issuing a SETPRT macro	EXLST operand and exit list
I/O error processing exit	When an I/O error has occurred	EXLST operand and exit list
JFCB	When opening a data set with OPEN TYPE=J and RDJFCB	EXLST operand and exit list
JFCBE	When opening a data set for the 3800	EXLST operand and exit list
Nonspecific tape volume mount	When a scratch tape is requested during OPEN or EOV routines	EXLST operand and exit list
QSAM parallel input	When opening a data set for QSAM processing	EXLST operand and exit list
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
User totaling area	When an I/O operation is scheduled and user totaling has been requested	EXLST operand and exit list
Volume security/ verification	When a scratch tape is requested during OPEN or EOV routines	EXLST operand and exit list

Figure 22 (Part 2 of 2). Data Management Exit Routines

For information on how to use the exits summarized in Figure 22 on page 73, see *Data Facility Product: Customization.*

Chapter 8. Spooling and Scheduling Data Sets

The job entry subsystem (JES) is a system function that spools and schedules input and output data streams.

Spooling includes two basic functions:

- Input streams are read from the input device and stored on an intermediate storage device in a format convenient for later processing by the system and by the user's program.
- Output streams are similarly stored on an intermediate device until a convenient time for printing or punching.

Scheduling provides the highest degree of system availability through the orderly use of system resources that are the objects of contention.

With spooling, unit record devices are used at full rated speed if enough buffers are available, and they are used only for the time needed to read, print, or punch the data. Without spooling, the device is occupied for the entire time that a job is doing other processing. Also, because data is stored instead of being transmitted directly, output can be queued in any order and scheduled by class and by priority within each class.

You enter data into the system input stream by preceding it with a DD * or a DD DATA JCL statement. This is a SYSIN data set.

Your output data can be printed or punched from a SYSOUT data set that is referred to as the output stream. You code the SYSOUT keyword parameter in your DD statement and designate the appropriate output class. For example, SYSOUT=A requests output class A. The class-device relationship is established for each installation, and a list of devices assigned to each output class will enable you to select the appropriate one. For further information on SYSIN and SYSOUT parameters, see *JCL*

SYSIN and SYSOUT must be either BSAM or QSAM data sets and you open and close them in the same manner as any other data set processed on a unit record device (except when multiple DCBs are used to write to the same output class, the records are not interspersed.) The DCB exit routine will be entered in the usual manner if you specify it in an exit list.

When you use QSAM with fixed-length blocked records or BSAM, the DCB block size parameter does not have to be a multiple of logical record length (LRECL) if the block size is specified through the SYSOUT DD statement. Under these conditions, if block size is greater than LRECL but not a multiple of LRECL, block size 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 block size without block size being a multiple of LRECL.

Because a SYSOUT data set is written on a direct access device, you should omit the DEVD operand in the DCB macro, or you should code DEVD=DA. Because SYSIN and SYSOUT data sets are spooled on intermediate devices, you should also avoid using device-dependent macros (such as FEOV, CNTRL, PRTOV, BSP, or SETPRT) in processing these data sets. (See "Device Independence" on page 56.) With a 3800, you can use SETPRT when processing spooled data sets. For further information, refer to *IBM 3800 Printing Subsystem Programmer's Guide.*

The job entry subsystem controls all blocking and deblocking of your data to optimize system operation and ignores the number of channel programs (NCP) you specify. The block size (BLKSIZE) and number of buffers (BUFNO) specified in your program have no correlation with what is actually used by the job entry subsystem. Therefore, you can select the blocking factor that best fits your application program with no effect on the spooling efficiency of the system. For QSAM applications, move mode is as efficient as locate mode.

All record formats are allowed, except that spanned records (RECFM=VS or VBS) cannot be specified for SYSIN. A record format of FIXED is supplied if it is not specified for SYSIN.

The minimum record length for SYSIN is 80 bytes. For undefined records, the entire 80-byte image is treated as a record. Therefore, a read of less than 80 bytes results in the transfer of the entire 80-byte image to the record area specified in the READ macro. For fixed and variable length records, an abend results if the LRECL is less than 80 bytes.

The logical record length value (JFCLRECL field in the JFCB) is filled in with the logical record length value of the input data set. This value is increased by 4 if the record format is variable (RECFM=V or VB). The logical record length may be a size other than the size of the input device, if the SYSIN input stream is supplied by an internal reader. The job entry subsystem will supply a value in the JFCLRECL field of the JFCB if that field is found to be zero.

The block size value (JFCBLKSI field in the JFCB) is filled in with the block size value of the input data set. This value is increased by 4 greater than the value calculated for the logical record value (that is, input data set logical record length + 4) if the record format is variable (RECFM=V or VB). The job entry subsystem will supply a value in the JFCBLKSI field of the JFCB if that field is found to be 0.

Your program is responsible for printing format, pagination, header control, and stacker select. You can supply control characters for SYSOUT data sets in the normal manner by specifying ANSI or machine characters in the DCB. Standard controls are provided by default if they are not specified. The length of output records must not exceed the allowable maximum length for the ultimate device. Cards can be punched in EBCDIC mode only.

Your SYNAD routine will be entered if an error occurs during data transmission to or from an intermediate storage device. Again, because the specific device is indeterminate, your SYNAD routine code should be device independent.

Chapter 9. Processing a Sequential Data Set

Data sets residing on any volume 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 includes data sets created using ISAM or a similar access method. Because the entire data set (prime, index, and overflow areas) will be processed, care should be taken to determine the type of records being processed.

Either the queued or the basic access technique may be used to store and retrieve the records of a sequential data set. For a non-DASD sequential data set, a technique called chained scheduling can be used to accelerate the input/output operations.

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.

In creating a sequential data set on a magnetic tape or direct access device, you must do the following:

- Code DSORG=PS or PSU in the DCB macro.
- Code a DD statement to describe the data set (see *JCL*).
- Process the data set with an OPEN macro (data set is opened for output or OUTIN), a series of PUT or WRITE and CHECK macros, and then a CLOSE macro.

Tape-to-Print, Move Mode—Simple Buffering: The example in Figure 23 on page 80 shows that the GET-move and PUT-move require two movements of the data records. If the record length (LRECL) does not change during 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.

```
           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
           LA       0,68(0,1)                    message address in regis-
*                                                ter 1.
           ST       14,SAVE14                    SYNAD routine prints part
           PUT      OUTDATA,(0)                  of the message (beginning
           SYNADRLS                              with the unit number) as
           L        14,SAVE14                    a 56-byte fixed-length
           RETURN                                record. It then returns
ENDJOB     CLOSE    (INDATA,,OUTDATA)            to the control 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 23. Creating a Sequential Data Set—Move Mode, Simple Buffering

Retrieving a Sequential Data Set

In retrieving a sequential data set on a magnetic tape or a direct access device, you must do the following:

- Code DSORG=PS or PSU in the DCB macro.
- Tell the system where your data set is located (by coding a DD statement; see *JCL*).
- Process the data set with an OPEN macro (data set is opened for input, INOUT, RDBACK, or UPDAT), a series of GET or READ macros, and then a CLOSE macro.

Tape-to-Print, Locate Mode—Simple Buffering: The example in Figure 24 on page 81 is similar to that in Figure 23. However, because 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
          MVI       78(1),C' '              Blank end of message
          MVC       79(49,1),78(1)
          ST        2,128(1)                Save address for debugging
          CH        0,=H'4'                 Test SYNADAF return code
          BE        MOVERCD                 Branch if data read
          BL        PRINTIT                 Branch if data not read
          CLI       128(1),C' '             See if data read anyway
          BE        PRINTIT                 Branch if definitely no data
MOVERCD   MVC       78(50,1),0(2)           Add input record to message
PRINTIT   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)
ERRORDCB  DCB       DDNAME=SYSOUTDD,DSORG=PS,MACRF=(PM),RECFM=V,         C
               BLKSIZE=128,LRECL=124
SAVE2     DS        F
          ...
```

Figure 24. Creating a Sequential Data Set—Locate Mode, Simple Buffering

Modifying a Sequential Data Set

You can modify a sequential data set in two ways:

- Changing the data in existing records (update in place)
- Adding new records to the end of a data set (extending the data set)

Updating a Sequential Data Set 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, GET, and PUTX macros.
- You cannot delete any record or change its length; you cannot add new records.
- The data set must be on a direct access device.

A record must be retrieved by a READ or GET macro before it can be updated by a WRITE or PUTX macro. A WRITE or PUTX macro does not need to be issued after each READ or GET macro. The READ and WRITE macros must be execute forms that refer to the same DECB; the DECB must be provided by the list forms of the READ or WRITE macros. (The execute and list forms of the READ and WRITE macros are described in *Data Administration: Macro Instruction Reference.*)

Updating with Overlapped Operations

To overlap input/output and processor activity, you can start several read or write operations before checking the first for completion. You cannot overlap read with 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 pending 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 42 on page 115, overlap is achieved by having a read or write request outstanding while each record is being processed. Note the use of the execute and list forms of the READ and WRITE macros, identified by the operands MF=E and MF=L.

Extending a Sequential Data Set

If you want to add records at the end of your data set, you must open the data set for output with DISP=MOD specified in the DD statement or specify the EXTEND option of the OPEN macro. You can then issue PUT or WRITE macros to the data set.

Concatenating Sequential Data Sets

Concatenation is a method by which a collection of sequential data sets can be treated by the application program as one data set. For example, two or more sequential data sets can be automatically retrieved by the system and processed successively as a single data set. 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 variation (such as record format, device, or option) makes them unlike.

Differing block sizes make data sets unlike unless the first block size in the concatenation is greater than any block size following. However, the system will automatically adjust buffers and channel programs for eligible data sets, if necessary. The primary purpose of this service is to permit otherwise homogeneous data sets to be concatenated in any order. Eligibility rules are:

- Data set resides on DASD
- RECFM cannot be U (RECFM can be F or V)
- Records are blocked
- LRECL is same for all data sets being concatenated
- Access method is QSAM
- OPEN got the buffer pool
- Block size was not coded in the DCB macro

Note: Block size can be coded in DD statements, via TSO ALLOCATE commands, or in the data set label.

When sequential data sets are concatenated, the system is open to only one of the data sets at a time. You must inform the system by modifying the DCBOFLGS field of the DCB if unlike sequential data sets are concatenated (this is not required for spool data sets, because EOV automatically treats them as unlike data sets). 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 Chapter 5, "Specifying a Data Control Block and Initializing Data Sets." 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 attributes during processing. When you concatenate data sets with unlike attributes, no EOV exits are taken.

When unlike sequential data sets are concatenated, the GET or READ macro instruction that detected the end of data set must be reissued. Figure 25 on page 84 illustrates a possible routine for determining when a GET or READ must be reissued. Also, you should not issue multiple input requests (that is, a series of READ or GET macros 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. These restrictions do not apply to like data sets, because no OPEN or CLOSE operation is necessary between data sets.

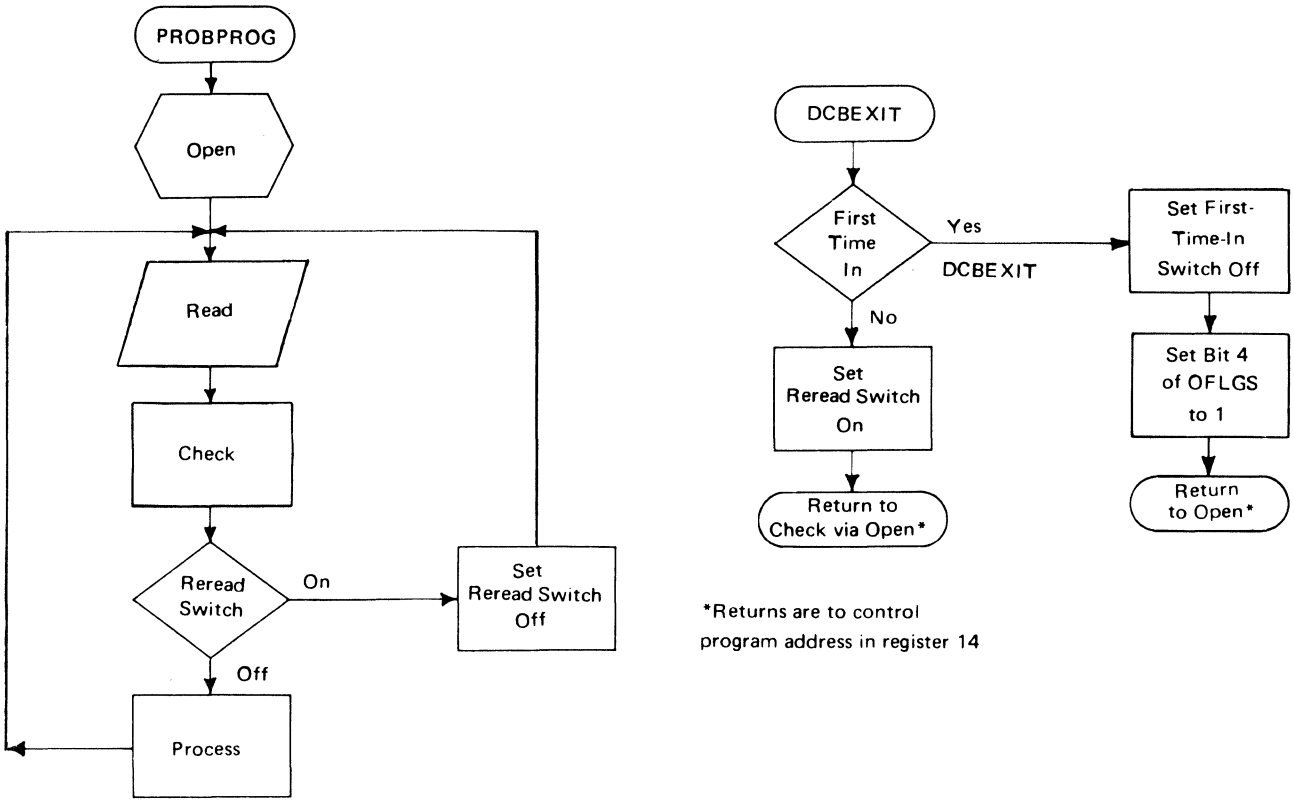


Figure 25. Reissuing a READ or GET for Unlike Concatenated Data Sets

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. Your end-of-data-set (EODAD) routine is not entered until the last data set has been processed.

To save time when processing two consecutive sequential data sets on a single tape volume, you specify LEAVE in your OPEN macro instruction. Concatenated data sets cannot be read backward.

Processing with Chained Scheduling

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. In a nonpageable (V=R) address space, the program-controlled interruption (PCI) flag in the CCWs cause the PCI appendage to get control and dynamically chain the next I/O request to the currently executing channel program.

The I/O performance is improved by reduction in both the processor time and the channel start/stop time required to transfer data within virtual storage. Some factors that affect performance improvement are:

- Address space type (real or virtual)
- BUFNO for QSAM
- The number of overlapped requests for BSAM (NCP)
- Other activity on the processor and channel

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 more buffers with QSAM, or must have a value of at least two and preferably more for NCP with BSAM or BPAM.

The system will default to chained scheduling for nondirect access devices (other than printers and format-U records on nondirect access devices), except for those cases in which it is not allowed.

A request for exchange buffering in MVS/XA is not honored, but compatibly defaults to move mode and therefore has no effect on either a request for chained scheduling or a default to chained scheduling.

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:

- CNTRL macro to be used.
- Bypassing of embedded DOS checkpoint records on tape input.
- Spooled data sets (SYSIN or SYSOUT).
- NCP=1 or BUFNO=1
- A print data set or any associated data set for the 3525 Card Punch. (For more information about programming the 3525, see *OS and OS/VS Programming Support for the IBM 3505 Card Reader and IBM 3525 Card Punch.*)

The number of channel program segments that can be chained is limited to the value specified in the NCP operand of BSAM DCBs, and to the value specified in the BUFNO operand of QSAM DCBs.

When the data set is a printer, chained scheduling is not supported (DCB=OPTCD=C) when channel 9 or channel 12 is in the carriage control tape or FCB.

When chained scheduling is being used, the automatic skip feature of the PRTOV macro for the printer will not function. Format control must be achieved by ANSI or machine control characters. (Control characters are discussed in more detail under "Carriage Control Character" on page 31, under "Record Format—Device Type Considerations" on page 28, and under Appendix B, "Control Characters" on page 183.) When you are using QSAM under chained scheduling to read

variable-length, blocked, ASCII tape records (format-DB), you must code BUFOFF=L in the DCB for that data set.

Note also that, if you are using BSAM with the chained scheduling option to read format-DB records and have coded a value for the BUFOFF operand other than BUFOFF=L, the input buffers will be converted from ASCII to EBCDIC as usual, but the record length returned to the DCBLRECL field will equal the block size, not the actual length of the record read in; the record descriptor word (RDW), if present, will not have been converted from ASCII to binary.

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 in a V=R region, separate channels should be assigned, if possible, when more than one data set is to be processed.

Note: Chained scheduling is not a DASD option; it is built into the access method for DASD.

Chained Scheduling Functions for DASD

For direct access storage devices (DASD), chained scheduling is not supported. (If the chained scheduling option is specified for DASD, it is ignored.) Instead, the functions of chained scheduling are performed directly by the sequential access method (either BSAM or QSAM).

In QSAM, the value of BUFNO determines how many channel programs or I/O requests will be chained together before I/O is initiated. The default value of BUFNO is 5; when five buffers are full (that is, five I/O requests have been issued), QSAM reads or writes all five records in a single revolution of the disk.

In BSAM, the first READ or WRITE instruction initiates I/O. Subsequent I/O requests (without an associated CHECK or WAIT instruction) will be put in a queue, and the channel program associated with the request will be chained to the previous request in the queue. During channel end processing for the first I/O request, the queue is checked for pending I/O requests and the next request in the queue is started using the channel end appendage. The number of I/O requests that may be chained together is limited to the number of requests that can be handled in one I/O event (and one revolution of the disk) before channel end processing is complete.

Search Direct for Input Operations

To accelerate the input operations required for a data set on DASD, the operating system uses a technique called search direct. Search direct reads in the requested record and the count field of the next record. This allows the operation to get the next record directly, along with the count field of the following record.

You may receive unpredictable results when your application has a dependency that is incompatible with the use of search direct. For example, you may receive unpredictable results when multiple DCBs are open for a file and one of the applications is adding records.

Determining the Length of a Record on Input

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

1. For fixed-length, unblocked records, the length of all records is the value in the DCBBLKSI field of the DCB.

2. For variable-length records, the block descriptor word in the record contains the length of the record.

3. For fixed-length blocked or undefined-length records, the following method can be used to calculate the block length. This method can be used with BSAM, BDAM, or BPAM. (This method should not be used when reading track overflow records or when using chained scheduling with format-U records. In these cases, the length of a record cannot be determined.) After checking the DECB for the READ request but before issuing any subsequent data management macros 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, 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, 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 26 on page 88.

```
          ....
          OPEN      (DCB,(INPUT))
          LA        DCBR,DCB
          USING     IHADCB,DCBR
          ...
          READ      DECB1,SF,DCB,AREA1,'S'
          READ      DECB2,SF,DCB,AREA2,50
          ...
          CHECK     DECB1
          LH        WORK1,DCBBLKSI          Block size 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           Block size 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 26. One Method of Determining the Length of the Record When Using BSAM to Read Undefined-Length Records

4. When an undefined-length record is read, the actual length of the record is returned in the DCBLRECL field of the data control block. Because of this use of DCBLRECL, the LRECL operand should be omitted. This method can only be used with QSAM and BSAM.

5. The length to be read or written can be supplied dynamically in a READ/WRITE macro using BSAM. This method cannot be used when using chained scheduling on any nondirect access device.

Writing a Short Block When Using the BSAM WRITE Macro

If you have fixed-blocked record format, you can change the length of a block when you are writing blocks for a sequential data set. The DCB block size field (DCBBLKSI) can be changed to specify a block size that is shorter than what was originally specified for the data set. The DCBBLKSI field must be changed before issuing the WRITE macro instruction and must be a multiple of the LRECL parameter in the DCB. Any subsequent WRITE macros issued will write records with the new block length until the block size is changed again. The DCB block size field should not be changed to specify a block size that is greater than what was originally specified for the data set.

Managing SAM Buffer Space

The operating system provides several methods of buffer acquisition and control. Each buffer (virtual 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 virtual storage area allocated for a given number of buffers of a given length.

The number of buffers you assign to a data set should be a trade-off against the frequency with which you refer to each buffer. A buffer that is not referred to for a relatively long period of time may be paged out. If this were allowed to happen to any considerable degree, it could decrease throughput.

Buffer segments and buffers within the buffer pool are controlled automatically by the system when the queued access technique is used. However, you can tell the system you are finished processing the data in a buffer by issuing a release (RELSE) macro for input or a truncate (TRUNC) macro instruction for output. The simple buffering technique can be used to process a sequential data set or 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 macros, or dynamically for BDAM and BISAM, by requesting dynamic buffering in your DCB macro instruction and your READ or WRITE macro. If you request dynamic buffering, the system will automatically provide a buffer each time a READ macro 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
- Explicitly, using the GETPOOL macro

- 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 virtual storage area must be returned to the system by the FREEPOOL macro.

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 ISCII/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 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, 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. The length of BUFL must be at least the block size.

When the data set using the buffer pool is closed, you can reuse the area as required. You can also reissue the BUILD macro 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 5 buffers (BUFNO=5), the BUILD macro should specify 10 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, like the BUILD macro, causes a buffer pool to be constructed in an area of virtual storage you provide. In addition, BUILDRCD makes it possible for you to access variable-length, spanned records as complete logical records, rather than as segments.

You must be processing with QSAM in the locate mode and you must be processing either VS/VBS or DS/DBS records, if you want to access the variable-length, spanned records as logical records. If you issue the BUILDRCD macro before the data set is opened, or during your DCB exit routine, you automatically get logical records rather than segments of spanned records.

Only one logical record storage area is built, no matter how many buffers are specified; therefore, you can't share the buffer pool with other data sets that may be open at the same time.

GETPOOL—Get a Buffer Pool

If a specified area is not reserved for use as a buffer pool, or if 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 causes the system to allocate a virtual storage area to a buffer pool. The system builds a buffer pool control block and stores its address in the data set's DCB. The GETPOOL macro should be issued either before opening of the data set or during your DCB's OPEN exit routine.

When using GETPOOL with QSAM, specify a buffer length (BUFL) at least as large as the block size.

Automatic Buffer Pool Construction

If you have requested a buffer pool and have not used an appropriate macro by the end of your DCB exit routine, the system automatically allocates virtual storage space for a buffer pool. The buffer pool control block is also assigned and the pool is associated with a specific DCB. For BSAM, a buffer pool is requested by specifying BUFNO. For QSAM, BUFNO can be specified or allowed to default to 5. 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, you should also issue a FREEPOOL or FREEMAIN macro (discussed in the next section).

FREEPOOL—Free a Buffer Pool

Any buffer pool assigned to a DCB either automatically by the OPEN macro (except when dynamic buffer control is used) or explicitly by the GETPOOL macro should be released before your program is terminated. The FREEPOOL macro should be issued to release the virtual storage area as soon as the buffers are no longer needed. When you are using the queued access technique, you must

close the data set first. If you are not using the queued access technique, it is still advisable to close the data set first.

If the OPEN macro was issued while running under a protect key of zero, a buffer pool that was obtained by OPEN should be released by issuing the FREEMAIN macro instead of the FREEPOOL macro. This is necessary because the buffer pool acquired under these conditions will be in storage assigned to subpool 252.

Constructing a Buffer Pool

Figure 27 and Figure 28 on page 93 illustrate several possible methods of constructing a buffer pool. They do not consider the method of processing or controlling the buffers in the pool.

In Figure 27, a static storage area named INPOOL is allocated during program assembly. The BUILD macro, issued during execution, arranges the buffer pool into 10 buffers, each 52 bytes long. Five buffers are assigned to INDCB and 5 to OUTDCB, as specified in the DCB macro 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. The 4-byte chain pointer that occupies the first 4 bytes of the buffer is included in the buffer, so no allowance need be made for this field.

```
            ...                         Processing
            BUILD     INPOOL,10,52         Structure a buffer pool
            OPEN      (INDCB,,OUTDCB,(OUTPUT))
            ...                            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 28 on page 93, two buffer pools are constructed explicitly by the GETPOOL macros. Ten input buffers are provided, each 52 bytes long, to contain one fixed-length record; 5 output buffers are provided, each 112 bytes long, to contain 2 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 macros. The same procedure should be used if the buffer pools were constructed automatically by the OPEN macro.

```
          ...
          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

Buffer Control

Your program can use any of 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 buffering is 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 three processing modes that determine the extent of data movement in virtual storage. Move, data, and locate mode processing can be specified for either the GET or PUT macro. (Substitute mode is no longer supported; the system defaults to move mode.) The movement of a record is determined as follows:

- *Move mode*— The record is moved from a system 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 that 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, the address returned in register 1 points to a record area where the spanned record is assembled or segmented.

  The PUT-locate routine uses the value in the DCBLRECL field to determine whether another record will fit into your buffer. Therefore, when you write a short record, you can maximize the number of records per block by modifying the DCBLRECL field before you issue a PUT-locate to get a buffer segment for the short record. The processing sequence follows:

1. Register 1 is returned to you with the address of the next buffer segment.
2. Move the record into the output buffer segment.
3. Put the length of the next (short) record into DCBLRECL.
4. Issue PUT-locate.
5. Move the short record into the buffer segment.

- *Substitute mode*— Move mode is used when substitute mode is requested in MVS/XA.

Two processing modes of the PUTX macro can be used in conjunction with a GET-locate macro. 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 virtual storage. The processing mode is specified by the operand of the PUTX macro, as explained in *Data Administration: Macro Instruction Reference*.

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

- Directly, using the GETBUF macro to retrieve a buffer constructed as described above. A buffer can then be returned to the pool by the FREEBUF macro.
- Dynamically, by requesting a dynamic buffer in your READ or WRITE macro. This technique can be used only when you are using BISAM or BDAM. If you request dynamic buffering, the system automatically provides a buffer each time a READ macro 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 without writing the record, the FREEDBUF macro 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.

Simple Buffering

The term simple buffering refers to the relationship of segments within the buffer. All segments in a simple buffer are together in 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 same 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 (step A, Figure 29 on page 96) 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 in register 0.

The PUT macro (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 can be used in place of the PUT-move macro. However, processing will be as described under "Exchange Buffering" (see PUT-substitute).

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

The GET macro 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 (step A, Figure 30 on page 96) specifies the address of a work area into which the system moves the next record from the input buffer.

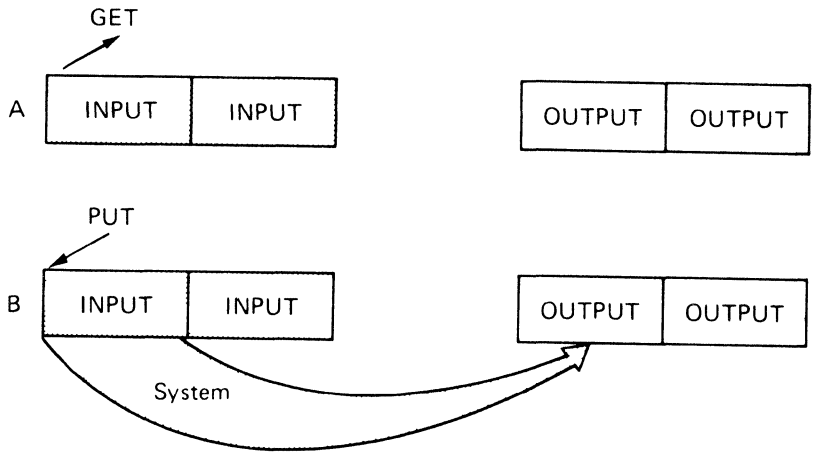


Figure 29. Simple Buffering with MACRF=GL and MACRF=PM

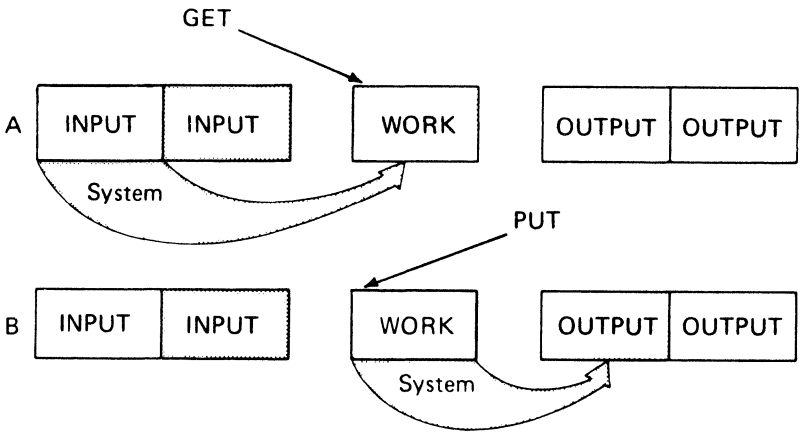


Figure 30. Simple Buffering with MACRF=GM and MACRF=PM

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

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

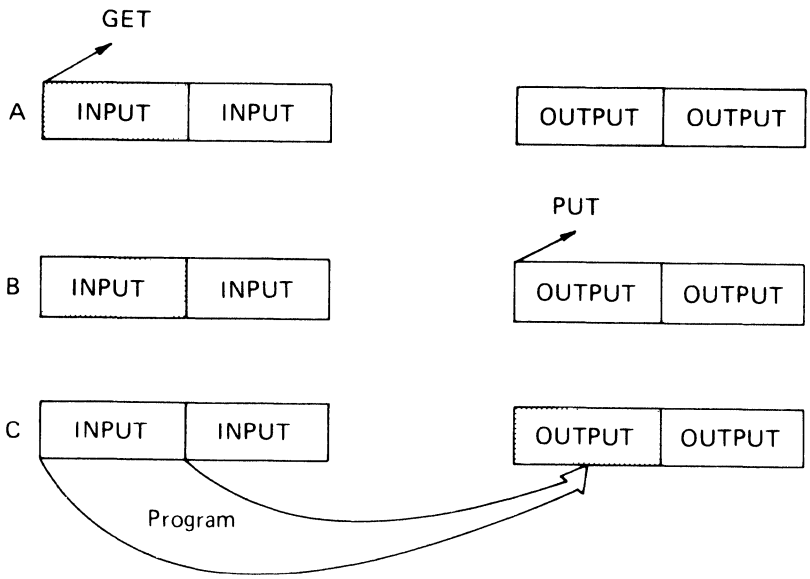
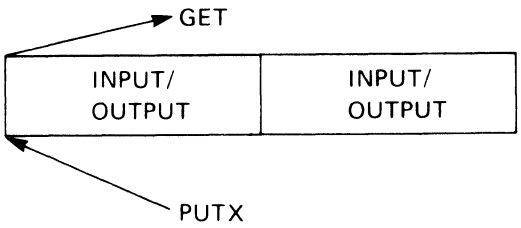


Figure 31. Simple Buffering with MACRF=GL and MACRF=PL

The PUT macro (step B, Figure 31) locates the address of the next available output buffer. Its address is returned in register 1. You must then move the record from the input buffer to the output buffer (step C, Figure 31). 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. The CLOSE and FEOV macros write the last record of your data set by issuing TRUNC and PUT macro instructions. Be careful not to issue an extra PUT before issuing CLOSE or FEOV. Otherwise, when the CLOSE or FEOV macro tries to write your last record, the extra PUT will write a meaningless record or produce a sequence error.

Simple Buffering—UPDAT Mode: When a data set is opened with UPDAT specified (Figure 32 on page 98), only GET-locate and PUTX-update are supported. The GET macro locates the next input record to be processed and its address is returned in register 1 by the system. The user may update the record and issue a PUTX macro that will cause the block to be written back in its original location in the data set after all the logical records in that block have been processed.



(No movement of data takes place)

Figure 32. Simple Buffering with MACRF=GL and MACRF=PM-UPDAT Mode

Exchange Buffering

Exchange buffering is not supported in MVS/XA. Its request is ignored by the system and move mode is used instead.

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 simple buffering. Figure 33 summarizes the combinations of buffering techniques and processing modes that can be used.

Input Buffering: Simple → / Actions ↓	GET-move, PUT-locate	GET-move, Put-move	GET-locate, PUT-locate	GET-locate, PUT-move	GET-locate (logical record), PUT-locate
Program must move record			X		X
System moves record	X	X		X	
System moves record segment					X
Work area required		X			
PUTX - output can be used				X	

Figure 33. Buffering Technique and GET/PUT Processing Modes

RELSE—Release an Input Buffer

When using the queued access technique to process a sequential or an indexed sequential data set, you can direct the system to ignore the remaining records in the input buffer being processed. The next GET macro 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 is issued. When you are using locate mode, the system does not refill the buffer until the next GET macro is issued. If a PUTX macro has been used, the block is written before the buffer is refilled.

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.

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 can be used with the basic access technique to request a buffer from a buffer pool constructed by the BUILD, GETPOOL, or OPEN macro. The address of the buffer is returned by the system in a register you specify when you issue the macro. If no buffer is available, the register contains a 0 instead of an address.

FREEBUF—Return a Buffer to a Pool

The FREEBUF macro is used with the basic access technique to return a buffer to the buffer pool from which it was obtained by a GETBUF macro. 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 need to free the buffer instead of writing the block, you must use the FREEDBUF macro. 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 is issued on the same DECB, unless you use the FREEDBUF macro.

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

Chapter 10. Processing a Partitioned Data Set

A partitioned data set is stored only on a direct access device. It is divided into sequentially organized members, each composed of one or more records (see Figure 34). 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 written or retrieved sequentially.

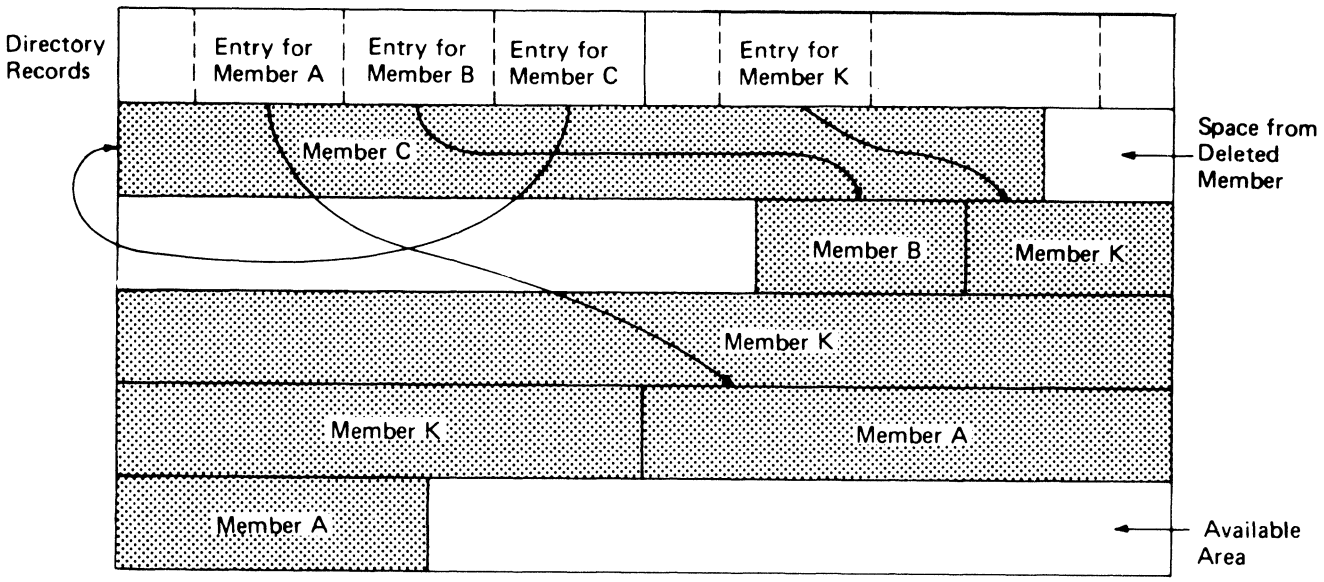


Figure 34. A Partitioned Data Set

The main advantage of using a partitioned data set is that, without searching the entire data set, you can retrieve any individual member after the data set is opened. For example, in a program library that is always a partitioned data set, each member 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; that is, compressed using the IEBCOPY utility.

The directory, a series of 256-byte 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 34. In addition, you can specify as many as 62 bytes of information in the entry. The directory entries are arranged by name in alphameric collating sequence.

The starting location of each member is recorded by the system as a relative track address (from the beginning of the data set) rather than as an absolute track address. Thus, an entire data set that has been compressed, 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 where 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, or no room on the volume for additional extents, no new members can be stored. A directory can not be extended and a partitioned data set may not cross a volume boundary.

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 BLDL, FIND, and STOW macros. 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 (including the count field). As shown in Figure 35, 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.

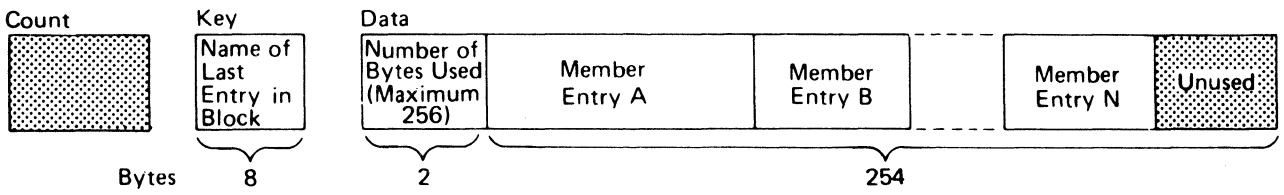


Figure 35. A Partitioned Data Set Directory Block

Each member entry contains a member name or an alias. Each entry also contains the relative track address of the member and a count field, as shown in Figure 36 on page 103. 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 1's.

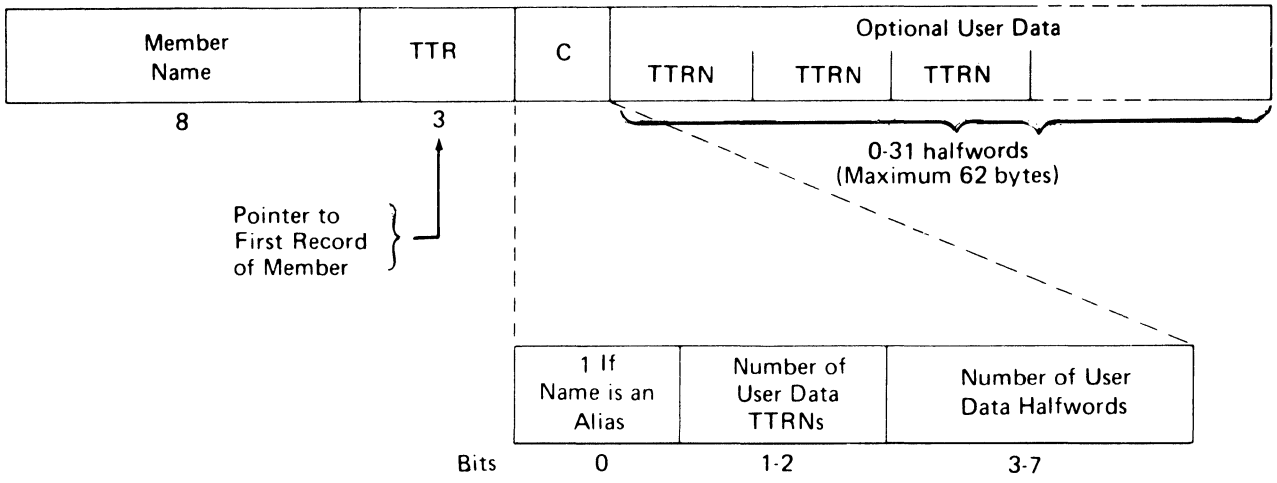


Figure 36. A Partitioned Data Set Directory Entry

NAME

specifies the member name or alias. It contains as many as 8 alphameric characters, left-justified, and padded with blanks if necessary.

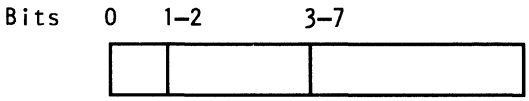
TTR

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.

The operating system supports a maximum of three pointers 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.

  Note: A note list for a partitioned data set containing variable length records does not conform to standard format.
- Variable-length records contain BDWs and RDWs that are treated as TTRXs by IEHMOVE.
- The pointers are not placed first in the user data field.
- Any direct access address (absolute or relative) is 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. 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:

User Data

TTRN	TTRN	TTRN	Optional

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 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:

TTRX

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. After checking the write operation, use the NOTE macro to determine the address of the list and place that address in the user data field of the directory entry.

Note: The linkage editor builds a note list for the load modules in overlay format. The addresses in the note list point to the overlay segments that are read into the system separately.

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 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 large, or if you expect to update it extensively, 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 re-creating the data set.

If the average member length is close to or less than the track length or if the track length exceeds 32760 bytes, the most efficient use of the direct access storage space may be made with a block size of 1/3 or 1/2 the track length. For example, BLKSIZE=23200 for a 3380 disk will yield two blocks per track. You might then ask for either 75 tracks, or 5 cylinders, thus allowing for 3480000 bytes of data.

Each member in a data set and each alias need one directory entry apiece. If you expect to have 10 members (10 directory entries) and an average of 3 aliases for each member (30 directory entries), allocate space for at least 40 directory entries.

Assuming an average length of 70000 bytes for each member, you need space for at least 50 directory entries. If each member also has an average of three aliases, space for an additional 150 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 200 directory entries, request at least 40 blocks. Any unused space on the last track of the directory is wasted unless there is enough space left to contain a block of the first member.

Either of the following space specifications would cause the same size allocation for a 3380 disk:

```
SPACE=(CYL,(5,,10))

SPACE=(TRK,(75,,10))
```

The following example would result in allocation of 100 tracks for data, plus 1 track for directory space:

```
SPACE=(23200,(100,,10))
```

Although a secondary allocation 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.

Creating a Partitioned Data Set

If you have no need to add entries to the directory, that is, the STOW macro will not be used, you can create a new data set and write the first member as follows (see Figure 37):

- Code DSORG=PS or DSORG=PSU in the DCB macro.
- 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, a series of PUT or WRITE macros, and then a CLOSE macro instruction. A STOW macro 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)),
               DISP=(NEW,CATLG),DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)---

         ...
         OPEN   (OUTDCB,(OUTPUT))
         ...
         PUT    OUTDCB,OUTAREA      Write record to member
         ...
         CLOSE  (OUTDCB)            Automatic STOW
         ...
OUTAREA  DS     CL80                Area to write from
OUTDCB   DCB    ---,DSORG=PS,DDNAME=PDSDD,MACRF=PM
```

Figure 37. 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 on the DD statement.

To take full advantage of the STOW macro, and thus the BLDL and FIND macros, in future processing, you can provide additional information with each directory entry. You do this by using the basic partitioned access technique, which also allows you to process more than one member without closing and reopening the data set, as follows (see Figure 38 on page 108).

- Request space in the DD statement for the entire data set and the directory.
- Define DSORG=PO or DSORG=POU in the DCB macro.
- 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, or to note the location of subgroups if there are any.
- 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. The STOW macro writes an end-of-file mark after the member.
- Continue to write, check, note, and stow until all the members of the data set and the directory entries have been written.

```
//PDSDD  DD    ---,DSN=MASTFILE,DISP=MOD,SPACE=(TRK,(100,5,7))

         ...
         OPEN  (OUTDCB,(OUTPUT))
         LA    STOWREG,STOWLIST   Load address of STOW list
         ...

**   WRITE MEMBER RECORDS AND NOTE LIST

MEMBER   WRITE DECBX,SF,OUTDCB,OUTAREA   WRITE  first record of member
         CHECK DECBX
         LA    NOTEREG,NOTELIST   Load address of NOTE list
*
         WRITE DECBY,SF,OUTDCB,OUTAREA   WRITE and CHECK next record
         CHECK DECBY
*
         NOTE  OUTDCB              To divide the member into subgroups,
         ST    R1,0(NOTEREG)       NOTE the TTRN of the first record in
*                                  the subgroup, storing it in the
*                                  NOTE list.
         LA    NOTEREG,4(NOTEREG)  Increment to next NOTE list entry
         ...
         WRITE DECBZ,SF,OUTDCB,NOTELIST   WRITE NOTE list record at the
*                                         end of the member
         CHECK DECBZ
         NOTE  OUTDCB               NOTE TTRN of NOTE list record
         ST    R1,12(STOWREG)       Store TTRN in STOW list
         STOW  OUTDCB,(STOWREG),A   Enter the information in directory
*                                   for this member after all records
*                                   and NOTE lists are written.
         LA    STOWREG,16(STOWREG)  Increment to the next STOW list entry
         ...
*
```

Repeat from label "MEMBER" for each additional member

```
*
         ...
         CLOSE (OUTDCB)       (NO automatic STOW)
         ...

OUTAREA  DS    CL80           Area to write from
OUTDCB   DCB   ---,DSORG=PO,DDNAME=PDSDD,MACRF=W
R1       EQU   1              Register one, return register from NOTE
NOTEREG  EQU   4              Register to address NOTE list
NOTELIST DS    0F             NOTE list
         DS    F              NOTE list entry  (4 byte TTRN)
         DS    19F            one entry per subgroup
STOWREG  EQU   5              Register to address STOW list
STOWLIST DS    0F             List of member names for STOW
         DC    CL8'MEMBERA'   Name of  member
         DS    CL3            TTR of first record (created by STOW)
         DC    X'23'          C byte, 1 user TTRN, 4 bytes of user data
         DS    CL4            TTRN of NOTE list
         ...                  one list entry per member (16 bytes each)
```

Figure 38. Creating Members of a Partitioned Data Set Using STOW

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 macros are provided by the operating system. The BLDL macro can be used to read one or more directory entries into virtual storage; the FIND macro locates a member of the data set and positions the DCB for subsequent processing; the STOW macro adds, deletes, replaces, or changes a member name in the directory. To use these macros, you must specify DSORG=PO or POU in the DCB macro. Before issuing FIND, BLDL, or STOW macro, you must check all preceding input/output operations for completion.

BLDL—Construct a Directory Entry List

The BLDL macro reads one or more directory entries into virtual storage. The member names are placed in a BLDL list that is constructed before the BLDL macro is issued. 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.

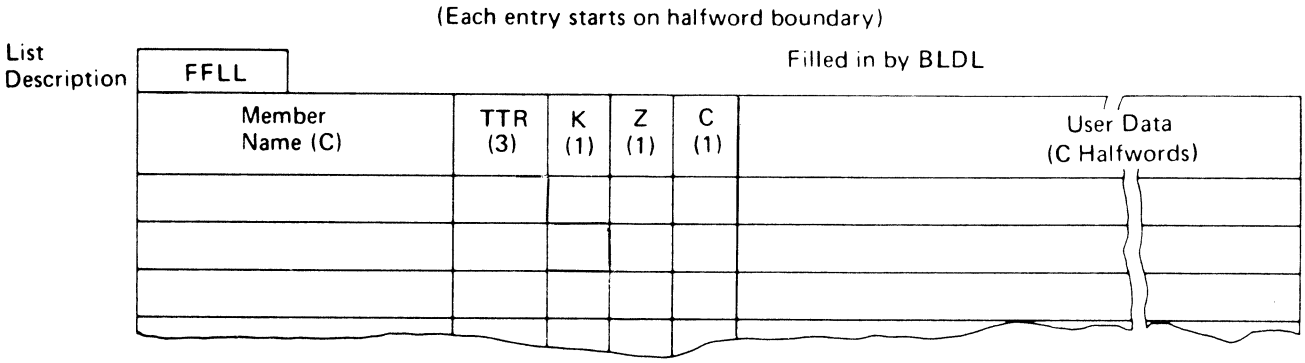
You can optimize retrieval time by directing a subsequent FIND macro instruction to the BLDL list rather than to the directory to locate the member to be processed.

The BLDL list, as shown in Figure 39 on page 110, 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 contain the TTR, K, Z, and C fields. 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, as many as 62 bytes can be reserved.

FIND—Position to a Member

To determine the starting address of a specific member, you must issue a FIND macro. 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. 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 BLDL list you have created by using the BLDL macro. 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 BLDL list entry.



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 single 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 39. BLDL List Format

The system will also search a concatenated series of directories when (1) a DCB is supplied that is opened for a concatenated partitioned data set or (2) a DCB is not supplied, in which case either JOBLIB or STEPLIB (themselves perhaps concatenated) followed by LINKLIB is searched.

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 begins processing at that point. You cannot use the FIND, BLDL, or STOW macro when you are processing one member as a sequential data set.

Because the DCBRELAD address in the data control block is updated when the FIND macro is used, you should not issue the FIND macro after WRITE and STOW processing without first closing the data set and reopening it for INPUT processing.

STOW—Update the Directory

When you add more than one member to a partitioned data set, you must issue a STOW macro after writing each member so that an entry for each one will be added to the directory. To use the STOW macro, DSORG=PO or POU must be specified in the DCB macro.

You can also use the STOW macro to delete, replace, or change a member name in the directory and store additional information with the directory entry. Because an alias can also be stored in the directory 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 alias entries or change them so that they point to the new member. An alias cannot be stored in the directory unless the member is present.

If you add only one member to a partitioned data set and indicate the member name in the DSNAME parameter of the DD statement, it is not necessary for you to use BPAM and a STOW macro in your program. If you want to do so, you may use BPAM and STOW, or BSAM or QSAM. If you use a sequential access method, or if you use BPAM and issue a CLOSE macro without issuing a STOW macro, the system will issue a STOW macro instruction using the member name you have specified on the DD statement. When the system issues the STOW, the directory entry that is added is the minimum length (12 bytes). This automatic STOW macro will not be issued if the CLOSE macro is a TYPE=T or if the TCB indicates the task is being abnormally terminated when the DCB is being closed. The DISP parameter on the DD statement determines what directory action parameter will be chosen by the system for the STOW macro.

If DISP=NEW or MOD was specified, a STOW macro with the add option will be issued. If the member name on the DD statement is not present in the data set directory, it will be added. If the member name is already present in the directory, the task will be abnormally terminated.

If DISP=OLD was specified, a STOW macro with the replace option will be issued. The member name will be inserted into the directory, either as an addition, if the name is not already present, or as a replacement, if the name is present.

Thus, with an existing data set, you should use DISP=OLD to force a member into the data set; you should use DISP=MOD to add members with protection against the accidental destruction of an existing member.

Retrieving a Member of a Partitioned Data Set

To retrieve a specific member from a partitioned data set, either the basic or the queued access technique can be used as follows (see Figure 40 on page 112):

- Code DSORG=PS or DSORG=PSU in the DCB macro.
- 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, a series of GET and READ macros, and then a CLOSE macro instruction.

```
//PDSDD  DD     ---,DSN=MASTFILE(MEMBERK),DISP=OLD
         ...
         OPEN   (INDCB)            Open for input, automatic FIND
         ...
         GET    INDCB,INAREA       Read member record
         ...
         CLOSE  (INDCB)
         ...
INAREA   DS     CL80               Area to read into
INDCB    DCB    ---,DSORG=PS,DDNAME=PDSDD,MACRF=GM
```

Figure 40. Retrieving One Member of a Partitioned Data Set

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

To process several members without closing and reopening, or to take advantage of additional data in the directory, this technique should be used (see Figure 41):

- Code DSORG=PO or POU in the DCB macro.
- Indicate in the DD statement the data set name of the partitioned data set by coding DSNAME=name and DISP=OLD.
- Issue the BLDL macro to get the list of member entries you need from the directory.
- Use the FIND or POINT macro 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.
- Your end-of-data-set (EODAD) routine receives control at the end of each member. At that time, you can process the next member or close the data set.
- Repeat this procedure for each member to be retrieved.

```
//PDSDD  DD    ---,DSN=MASTFILE,DISP=OLD

         ...
         OPEN  (INDCB)             Open for input, no automatic FIND
         ...
         LA    BLDLREG,BLDLLIST    Load address of BLDL list
         BLDL  INDCB,BLDLLIST      Build a list of selected member
*                                  names in virtual storage
         LA    BLDLREG,4(BLDLREG)  Point to the first entry
         ...
```

Read the NOTE list

```
MEMBER   LA    NOTEREG,NOTELIST    Load address of NOTE list
         MVC   TTRN(4),14(BLDLREG)  Move NOTE list TTRN
*                                  to fullword boundary
         POINT INDCB,TTRN          Point to the NOTE list record
         READ  DECBX,SF,INDCB,(NOTEREG)     Read the NOTE list
         CHECK DECBX
         ...
```

Read data from a subgroup

```
SUBGROUP POINT INDCB,(NOTEREG)     Point to subgroup
         READ  DECBY,SF,INDCB,INAREA  Read record in subgroup
         CHECK DECBY
         LA    NOTEREG,4(NOTEREG)  Increment to next subgroup TTRN
         ...
```

Repeat from label "SUBGROUP" for each additional subgroup
Repeat from label "MEMBER" for each additional member

```
         ...
         CLOSE (INDCB)
         ...

INAREA   DS    CL80
INDCB    DCB   ---,DSORG=PO,DDNAME=PDSDD,MACRF=R
TTRN     DS    F              TTRN of the NOTE list to point at
NOTEREG  EQU   4              Register to address NOTE list entries
NOTELIST DS    0F             NOTE list
         DS    F              NOTE list entry  (4 byte TTRN)
         DS    19F            one entry per subgroup
BLDLREG  EQU   5              Register to address BLDL list entries
BLDLLIST DS    0F             List of member names for BLDL
         DC    H'10'          Number of entries (10 for example)
         DC    H'18'          Number of bytes per entry
         DC    CL8'MEMBERA'   Name of  member
         DS    CL3            TTR of first record (created by BLDL)
         DS    X              K byte, concatenation number
         DS    X              Z byte, location code
         DS    X              C byte, flag and user data length
         DS    CL4            TTRN of NOTE list
         ...                  one list entry per member (18 bytes each)
```

Figure 41. Retrieving Several Members and Subgroups of a Partitioned Data Set

Modifying a Partitioned Data Set

Updating a Member of a Partitioned Data Set

A member of a partitioned data set can be updated in place, or it 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 macros.
- You cannot update concatenated partitioned data sets.
- 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 before it can be updated by a WRITE macro. Both macros 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 macros are described in *Data Administration: Macro Instruction Reference.*)

Updating with QSAM: You can update a member of a partitioned data set using the locate mode of QSAM (DCB specifies MACRF=PL) and using the PUTX macro. The DD statement must specify the data set and member name in the DSNAME parameter. This method allows only the updating of the member specified in the DD statement.

Updating with Overlapped Operations: To overlap input/output and processor 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 outstanding 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 42 on page 115, overlap is achieved by having a read or write request outstanding while each record is being processed. Note the use of the execute and list forms of the READ and WRITE macros, identified by the operands MF=E and MF=L.

```
//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
          WRITE   (2),SF,MF=E                       (R3) Write updated R2
```

(If R3 requires an update, branch to R3UPDATE)

```
          CHECK   (2)                               If R3 requires no update,
          B       READRECD                          check 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 42. Updating a Member of a Partitioned Data Set

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. Because 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.

When you rewrite the member, you must provide two DCBs, one for input and one for output. Both DCB macros 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. Although the old member is, in effect, deleted, its space cannot be reused until the data set is reorganized.

If an out-of-space condition occurs when updating a PDS member, the error recovery procedure will STOW the PDS member as 'TEMPNAME'. The original member will remain intact.

Processing a Partitioned Data Set Residing on MSS

If OPTCD=H is specified in the DCB subparameter of a DD statement, it specifies that, if a partitioned data set is being opened for input and resides on an MSS device, then at OPEN time the data set is staged to EOF on the virtual DASD device. If the option is not specified, only the directory is staged at OPEN time and cylinder faults occur during processing. This option might be used with the IEBCOPY utility program opening the PDS to reorganize and compress the data space. This BPAM option, OPTCD=H, may only be coded on the DD statement.

Concatenating Partitioned Data Sets

Two or more 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. 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.

Partitioned Concatenation

When partitioned data sets are concatenated, the system treats the group as a single data set and only one data extent block (DEB) is constructed. The maximum number of partitioned data sets that can be concatenated is 123 extents (input data sets only). For example, 123 single extent data sets can be concatenated but 8 data sets each with 16 extents cannot be concatenated.

Concatenated partitioned data sets are always treated as having like attributes and use the attributes of the first data set only.

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 to begin processing a member; you cannot use the POINT (or NOTE) macro until after the FIND macro has been issued. Figure 41 on page 113 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 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 provides the concatenation number of the data set to which the member belongs in the K field of the BLDL list. (See "BLDL—Construct a Directory Entry List" on page 109.)

Reading a BPAM Directory Sequentially

You can read a BPAM directory sequentially just by opening the data set to its beginning (without using positioning macros) and reading it.

- The DD statement should identify the DSNAME without a member name. You should specify a disposition option of either OLD or SHR.
- You can use either BSAM or QSAM with MACRF=R or G.
- Specify BLKSIZE=256 and RECFM=F.
- You must test for the last directory entry (X'FFFFFFFF').
- If you also want to read the keys (the name of the last member in that block), use BSAM and specify KEYLEN=8.

Chapter 11. 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.

The DSORG parameter of the DCB macro specifies the type of processing to be performed; DSORG in the DD statement specifies the organization of the data set when it is created.

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 virtual 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 LRECL field is not used when processing a direct data set. Only BLKSIZE must be specified when you read, add, or update records on a direct data set.

If dynamic buffering is specified for your direct data set, the system will provide a buffer for your records. If dynamic buffering is not specified, you must provide a buffer for the system to use.

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.

Direct Data Set Organization

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. Therefore, each record should be assigned a unique key. If identical keys are used, it is possible, during periods of high processor and channel activity, to skip the desired record and retrieve the next record on the track. 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 as a relative block number, 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. Because 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.

Creating a Direct Data Set

After 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 BSAM DCB macro should be used with the WRITE macro instruction (the form used to create a direct data set). The following parameters must be specified in the DCB macro instruction:

- DSORG=PS or PSU
- DEVD=DA or omitted
- MACRF=WL

The DD statement must indicate direct access (DSORG=DA or DAU). If keys are used, a key length (KEYLEN) must also be specified. Record length (LRECL) need not be specified but may be used to provide compatibility with sequential access method processing of this data set.

It is possible to create a direct data set using QSAM (no keys allowed) or BSAM (with or without keys and the DCB specifies MACRF=W). However, this method is not recommended because, when you access this direct data set, you cannot request a function that requires the information in the capacity record (R0) data field. For example, the following restrictions would apply:

- Variable-length, undefined-length, or variable-length spanned record processing is not allowed.
- The WRITE add function with extended search for fixed-length records (with or without track overflow) is not allowed.

If a VIO data set is opened for processing with the extended search option, the DEBENDCC and DEBENDHH fields of the DEB will reflect the real address of the last record written during the BDAM create step. This prevents BDAM from

searching unused tracks. The information needed to determine the data set size is written in the DSCB during the close of the DCB used in the create step. Therefore, if this data set is being created and processed by the same program, and the DCB used for creating the data set has not been closed before opening the DCB to be used for processing, the resultant beginning and ending CCHH will be equal.

If a direct data set is created and updated or read within the same job step, and the OPTCD parameter is used in the creation, updating, or reading of the data set, different DCBs and DD statements should be used.

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 for format-U, format-V, or format-VS records, write a capacity record. The capacity record (R0) contains a 7-byte data field (CCHHRLL), where CCHHR is the ID of the last record on the track, and LL is the number of unused bytes on the track. If a WRITE SZ macro is issued for a track with no records, R is zero and LL is the entire length of the track.

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 43 on page 122, a tape containing 204-byte 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     8F
DALOAD   DCB    DSORG=PS,MACRF=(WL),DDNAME=DAOUTPUT,          C
                DEVD=DA,SYNAD=CHECKER,---
TAPEDCB  DCB    EODAD=INPUTEND,MACRF=(GL), ---
         ...
```

Figure 43. Creating a Direct Data Set

Referring to a Record in a Direct Data Set

After 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. 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 or Block Address and Actual Key: In addition to the relative track or block address, you specify the address of a virtual 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 where a record can be added.

To use the extended search option, you must indicate in the DCB (DCBLIMCT) 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: When more than one task is referring to the same data set, exclusive control of the block being updated is required to prevent simultaneous reference to the same record. Rather than issuing an ENQ macro each time you update a block, you can request exclusive control through the MACRF field of the DCB and the type operand of the READ macro. The coding example in Figure 45 on page 126 illustrates the use of exclusive control. After the READ macro is executed, your task has exclusive control of the block being updated. No other task in the system requesting access to the block is given access until the operation started by your WRITE macro is complete. If, however, the block is not to be written, you can release exclusive control using the RELEX macro.

Feedback Option: This option specifies that the system is to provide the address of the record requested by a READ or WRITE macro. This address may be in the same form that was presented to the system in the READ or WRITE macro, or as an 8-byte actual address. This option can be specified in the OPTCD parameter of the DCB and in the READ or WRITE macro. If this option is omitted from the DCB but is requested in a READ or WRITE macro, an 8-byte actual address is returned to the user.

The feedback option is automatically provided for a READ macro instruction requesting exclusive control for updating. This feedback will be in the form of an actual address (MBBCCHHR) unless feedback was specified in the OPTCD field of the DCB. In this case, feedback is returned in the format of the addressing scheme used in the problem program (an actual or a relative address). When a WRITE or RELEX macro is issued (which releases the exclusive control that was gotten for the READ request), the system will assume that the addressing scheme used for the WRITE or RELEX macro is in the same format as the addressing scheme used for feedback in the READ macro.

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 want to add a record passing a relative block address, the system converts the address to an actual 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 found on that track or the first within the search limits, if requested. If extended search is used, the search begins with the first record on the track. Without extended search, the search may start at any record on the track. Therefore, records that were added to a track are not necessarily located on the track in the same sequence they were written in.

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 (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 access the record only by its relative track or actual address (direct addressing only). When you add a record to this data set, you must retain the relative track or actual address data (for example, by updating your cross-reference table). The extended search option is not allowed because it requires keys.

Tape-to-Disk Add—Direct Data Set: The example in Figure 44 on page 125 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 instead of the CHECK macro to test for errors or exceptional conditions.

```
//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 44. Adding Records to a Direct Data Set

Tape-to-Disk Update—Direct Data Set: The example in Figure 45 is similar to that in Figure 44, but involves updating rather than adding. There is no check for dummy records. The existing direct data set contains 25000 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 virtual storage for updating.

When you are updating variable-length records, you should use the same length to read and write a record.

```
//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,DIX,DIRECT,'S','S',0,REF+1
         CHECK    DECBRD
         L        3,DECBRD+12
         MVC      0(30,3),DATA
         ST       3,DECBWR+12
         WRITE    DECBWR,DIX,DIRECT,'S','S',0,REF+1
         CHECK    DECBWR
         B        NEXTREC
         ...
KEYFIELD DS       0D
         DC       XL3'0'
KEY      DS       CL5
DATA     DS       CL30
REF      DS       F
DIRECT   DCB      DSORG=DA,DDNAME=DIRECTDD,MACRF=(RISXC,WIC),       C
             OPTCD=RF,BUFNO=1,BUFL=100
TAPEDCB  DCB      ---
         ...
```

Figure 45. Updating a Direct Data Set

Consideration for User Labels: User labels, if wanted, 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.

Consideration for using the 2305-2 Fixed Head Storage: When a data set on a 2305-2 device is to be used by several tasks simultaneously, or when overlapping I/O (successive writes issued without an intervening CHECK or WAIT) is used, the following combination may produce overlaying of records:

- WRITE-add processing
- Fixed records with or without track overflow

Sharing Direct Data Sets

BDAM permits several tasks to share the same DCB and several jobs to share the same data set. It synchronizes I/O requests at both levels by maintaining a read-exclusive list.

When several tasks share the same DCB and each asks for exclusive control of the same block, BDAM issues a system ENQ for the block (or in some cases the whole track). It reads in the block and passes it to the first caller while putting all subsequent requests for that block on a wait queue. When the first task releases the block, BDAM moves it into the next caller's buffer and posts it complete. The block is passed to subsequent callers in the order the request was received.

BDAM not only synchronizes the I/O requests, but also issues only one ENQ and one I/O request for several read requests for the same block.

Note: Because BDAM processing is not sequential and I/O requests are not related, a caller can continue processing other blocks while waiting for exclusive control of the shared block.

Because BDAM issues a system ENQ for each record held exclusively, it allows a data set to be shared between jobs, so long as all callers use BDAM. BDAM's commonly understood argument is what is enqueued on.

BDAM supports multiple task users of a single DCB when working with existing data sets. When operating in load mode, however, only one task may use the DCB at a time. The following restrictions and comments apply when more than one task shares the same DCB, or when using multiple DCBs for the same data set.

- Subpool 0 must be shared.
- The user should ensure that a WAIT or CHECK macro has been issued for all outstanding BDAM requests before the task issuing the READ or WRITE macro terminates. In case of abnormal termination, this can be done through a STAE/STAI or ESTAE exit.
- FREEDBUF and/or RELEX macros should be issued to free any resources that could still be held by the terminating task. This can be done during or after task termination.

Note: Open, close, and all I/O must be performed in the same key and state (problem state or supervisor state).

Chapter 12. 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 as many as 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 you know the name of the book or the author, you can look in the card catalog and obtain a catalog number that will enable you to locate the book in the book files. You then go to the shelves and proceed through rows until you find 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 you proceed through the rows, you compare the catalog number obtained from the index with the numbers posted on each row. Upon locating the proper row, you search that row for the shelf that contains the book. Then you look at the individual book numbers on that shelf until you find 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 46.

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 last track of prime data is reserved for an end-of-file mark. The portion of Figure 46 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.

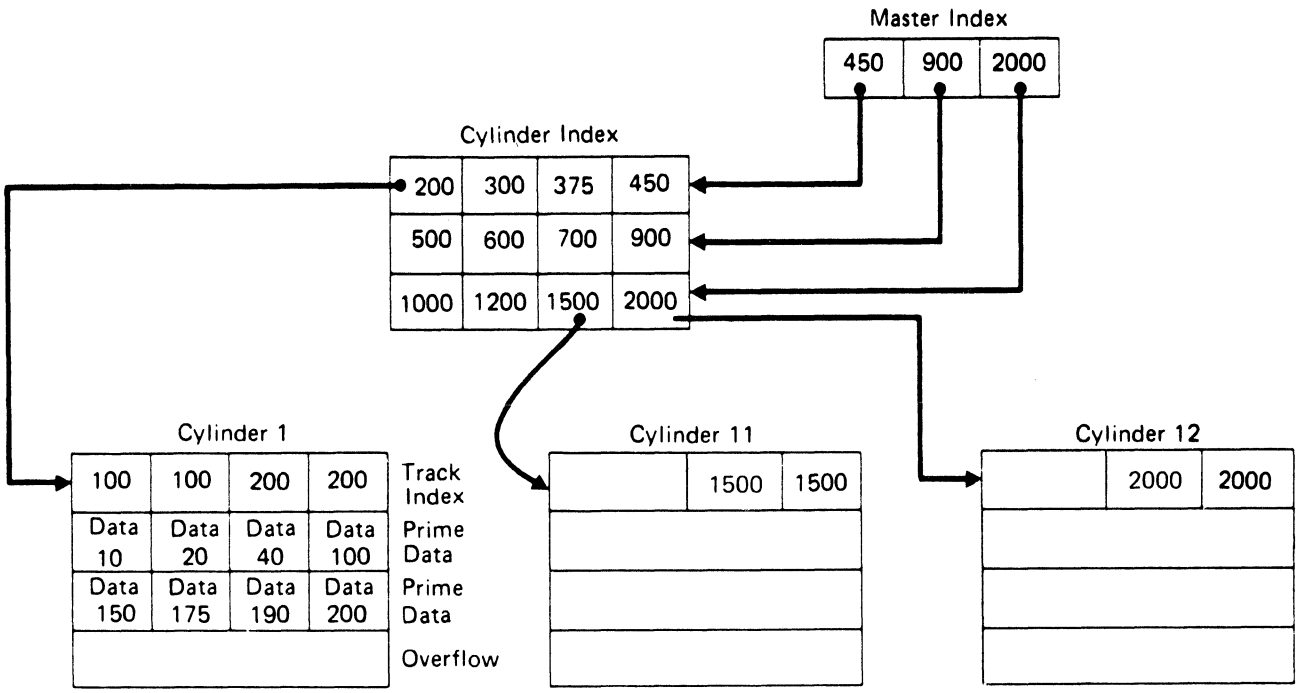


Figure 46. Indexed Sequential Data Set Organization

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

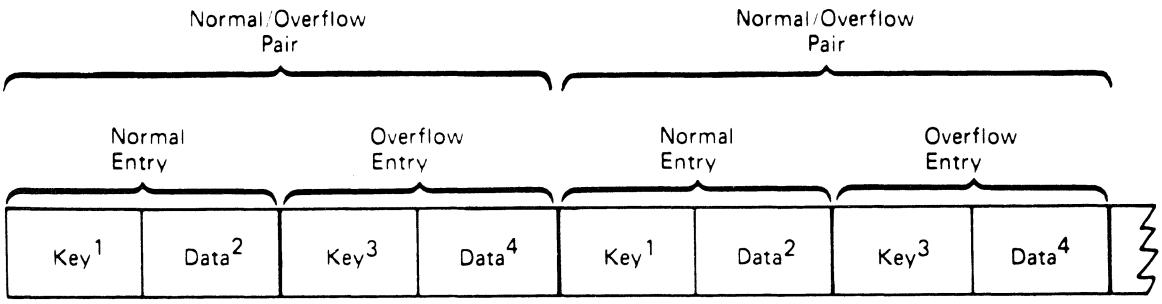
The operating system generates track and cylinder indexes automatically. As many as three levels of master index are created if requested.

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 (a track shared by index and data). 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 46.) 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 47 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 of each cylinder 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 47. Format of Track Index Entries

Each index entry has the same format as the others. 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, 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 that points to a track index. Because 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. The presence of this index makes long, serial searches through a large, cylinder index unnecessary.

You can specify the conditions under which you want a master index created. For example, if you have specified NTM=3 and OPTCD=M in your DCB macro, a master index is created when the cylinder index exceeds 3 tracks. The master index consists of one entry for each track of cylinder index. If your data set is extremely large, a higher-level master index is created when the first-level master index exceeds three tracks. This higher-level master index consists of one entry for each track of the first-level master index. This procedure can be repeated for as many as three levels of master index.

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.

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 called 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 DSORG=ISU and MACRF=PM or MACRF=PL in the DCB macro.
- Specify in the DD statement the DCB attributes DSORG=IS or DSORG=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 to place all the records or blocks on the direct access volume.
- Close the data set.

The records that comprise 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 an unload is done that deletes all existing records in an ISAM data set, at least one record must be written on the subsequent load. If no record is written, the data set will be unusable.

If the records are blocked, you should not write a record with a hexadecimal value of FF and a key of hexadecimal value FF. This value is used for padding. If it occurs as the last record of a block, the record cannot be retrieved. If the record is moved to the overflow area, it is lost.

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 virtual storage. Note that the full-track-index write option can be used only for fixed-length records.

If you do not specify full-track-index write, 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 do specify full-track-index write, the operating

system accumulates track index entries in virtual storage until either (a) there are enough entries to fill a track or (b) 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. This option requires allocation of more storage space (the space in which the track index entries are gathered), but the number of I/O operations required to write the index can be significantly decreased.

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

After an indexed sequential data set has been created, its characteristics cannot be changed. However, for added flexibility, the system allows you to retrieve records by 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 48 on page 135 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 through 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 6 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.

```
//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=3330,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 48. Creating an Indexed Sequential Data Set

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.

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 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 macros 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 re-create 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 system that includes resume load. This is necessary 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. Resume load for variable-length records begins at the next sequential track of the prime data set. 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 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. However, if the SYNAD exit is taken during CLOSE, register 0 will contain the IOB address.

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 and must be of the same device type as the prime area. 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. 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:

```
Overflow 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 cylinder for the end-of-file filemark. You may not request an independent index for an ISAM data set that has only 1 cylinder of prime data.

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 49 on page 137.

- Space can be requested only in cylinders, SPACE=(CYL,(...)), or absolute tracks, SPACE=(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 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=(3380,3),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.

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 *Utilities*.

Criteria 1. Number of DD Statements	2. Types of DD Statements	3. Index Size Coded?	Restrictions on Unit Types and Number of Units Requested	Resulting Arrangement of Areas
3	INDEX PRIME OVFLOW	—	None	Separate index, prime, and overflow areas.
2	INDEX PRIME	—	None	Separate index and prime areas. Any partially used index cylinder is used for independent overflow if the index and prime areas are on the same type of device.
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 partially used index cylinder is used for independent overflow.
1	PRIME	Yes	Statement cannot request more than one unit.	Prime area with embedded index area; independent overflow in remainder of partially used index cylinder.

Figure 49. Requests for Indexed Sequential Data Sets

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,
//          SPACE=(CYL,quantity,,CONTIG),UNIT=unitname,
//          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. When you make the computations, consider how much additional space is required for device overhead. Figure 58 on page 189 and Figure 59 on page 190 show device capacities and overhead formulas. In the formulas that follow, the length of the last (or only) block, shown below as Bn, must include device overhead as given in Figure 59.

```
Blocks =  Track capacity / Length of blocks
```

The following eight steps summarize calculation of space requirements for an indexed sequential data set.

Note: Use modulo-32 arithmetic when calculating key length and data length terms in your equations. Compute these terms first, then round up to the nearest increment of 32 bytes before completing the equation.

Step 1: After 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          = (Maximum number of blocks
tracks required      / Blocks per track) + 1
```

ISAM load mode reserves the last prime data track for the file mark.

Example: Assume that a 200000 record parts-of-speech dictionary is stored on an IBM 3380 Disk Storage 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 parts-of-speech code and control information. Each block contains 50 records; LRECL=20 and BLKSIZE=1000. Using the formula from Figure 59 on page 190, we find that each track will contain 26 blocks or 1300 records. A total of 155 tracks will be required for the dictionary.

```
Blocks = 47968/(256+((12+267)/32)(32)+((1000+267)/32)(32))
       = 47968/1824 = 26

Records per track = (26 blocks)(50 records per block) = 1300

Prime data
tracks      = (200000 records / 1300 records per track) + 1 = 155
required
```

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 also that, 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.

```
Overflow records = Track capacity / Length of overflow records
per track
```

Example: Approximately 5000 overflow records are expected for the data set described in step 1. Because 55 overflow records will fit on a track, 91 overflow tracks are required. These are 91 overflow tracks for 155 prime data tracks, or approximately 1 overflow track for every 2 prime data tracks. Because the 3380 disk pack has 15 tracks per cylinder, it would probably be best to allocate 5 tracks per cylinder for overflow.

```
Overflow  = 47968/(256+((12+267)/32)(32)+((30+267)/32)(32))
records   = 47968/864
per track = 55

Overflow  = 5000 records / 55 records per track
tracks    = 91
required

Overflow tracks per cylinder = 5
```

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?

```
Index entries = Track capacity / Length of index entries
per track
```

Example: Again assuming a 3380 disk pack and records with 12-byte keys, we find that 59 index entries fit on a track.

```
Index     = 47968/(256+((12+267)/32)(32)+((10+267)/32)(32))
entries   = 47968/832
per track = 57
```

Step 4: Unused space on the last track of the track index is a function of the number of tracks required for track index entries, which in turn depends upon the number of tracks per cylinder and the number of track index entries per track. You can use any unused space for any prime data records that will fit.

```
Unused     = (Number of index entries per track)
space        - (2 (Number of tracks per cylinder
             - Number of overflow tracks per cylinder) + 1)
               (Number of bytes per index)
```

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 this case, if the remainder of the division is less than or equal to 2, drop the remainder. In all other cases, round the quotient up to the next integer.

Example: The 3380 disk pack has 15 tracks per cylinder. You can fit 57 track index entries into one track. Therefore, you need less than 1 track for each cylinder.

```
Number of
trk index = (2 (15 - 5) + 1) / (57 + 2)
trks per  = 21 / 59
cylinder
```

The space remaining on the track is 47968 - (21 (832)) = 30496 bytes.

This is enough space for 16 blocks of prime data records. Because the normal number of blocks per track is 26, the blocks use 16/26ths of the track, and the effective number of track index tracks per cylinder is therefore 1 - 16/26 or 0.385.

Note that space is required on the last track of the track index for a dummy entry to indicate the end of the track index. The dummy entry consists of an 8-byte count field, a key field the same size as the key field in the preceding entries, and a 10-byte data field.

Step 5: Next you have to calculate the number of tracks available on each cylinder for prime data records. You cannot include tracks set aside for cylinder overflow records.

```
Prime data  = Tracks per cylinder
tracks per    - Overflow tracks per cylinder
cylinder      - Index tracks per cylinder
```

Example: If you set aside 5 cylinder overflow tracks, and you need 0.385ths of a track for the track index, 9.615 tracks are available on each cylinder for prime data records.

```
Prime data tracks  = 15 - 5 - (0.385) = 9.615
per cylinder
```

Step 6: The number of cylinders required to allocate prime space is determined by the number of prime data tracks required divided by the number of prime data tracks available on each cylinder. This area includes space for the prime data records, track indexes, and cylinder overflow records.

```
Number of  = Prime data tracks needed
cylinders    / Prime data tracks per cylinder needed
needed
```

Example: You need 155 tracks for prime data records. You can use 9.615 tracks per cylinder. Therefore, you need 17 cylinders for your prime area and cylinder overflow areas.

```
Number of   = (155) / (9.615) = 16.121   (round up to 17)
cylinders
required
```

Step 7: You will need space for a cylinder index and 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  = (Track indexes + 1)
required for      / (Index entries per track cylinder index)
```

Example: You have 17 track indexes (from Step 6). Because 57 index entries fit on a track (from Step 3), you need 1 track for your cylinder index. The remaining space on the track is unused.

```
Number of tracks required = (17 + 1) / 57 = 18 / 57 = 0.316 < 1
for cylinder index
```

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 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. As many as 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.

Calculate the number of tracks for master indexes as follows:

```
# Tracks for master indexes =
(# Cylinder index tracks + 1) / Index entries per track
```

If the number of cylinder indexes is greater than NTM, calculate the number of tracks for a first level master index as follows:

```
# Tracks for first level master index =
(Cylinder track indexes + 1) / Index entries per track
```

If the number of first level master indexes is greater than NTM, calculate the number of tracks for a second level master index as follows:

```
# Tracks for second level master index =
(First level master index + 1) / Index entries per track
```

If the number of second level master indexes is greater than NTM, calculate the number of tracks for a third level master index as follows:

```
# Tracks for second level master index =
(Second level master index + 1) / Index entries per track
```

Example: Assume that your cylinder index will require 22 tracks. Because 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.

```
Number of tracks required  = (22 + 1) / 10 = 2.3
for master indexes
```

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?

   ```
   Blocks = Track capacity / Length of blocks
   ```

2. How many overflow records will fit on a track?

   ```
   Overflow records = Track capacity
                    / Length of overflow records per track
   ```

3. How many index entries will fit on a track?

   ```
   Index entries  = Track capacity / Length of index entries
   per track
   ```

4. How much space is left on the last track of the track index?

   ```
   Unused     = (Number of index entries per track)
   space      - (2 (Number of tracks per cylinder
              - Number of overflow tracks per cylinder) + 1)
                (Number of bytes per index)
   ```

5. How many tracks on each cylinder can you use for prime data records?

   ```
   Prime data  = Tracks per cylinder
   tracks per  - Overflow tracks per cylinder
   cylinder    - Index tracks per cylinder
   ```

6. How many cylinders do you need for the prime data area?

   ```
   Number of  = Prime data tracks needed
   cylinders  / Prime data tracks per cylinder needed
   needed
   ```

7. How many tracks do you need for the cylinder index?

```
Number of tracks
required for      = (Track indexes + 1)
cylinder index    / Index entries per track
```

8. How many tracks do you need for master indexes?

```
Number of tracks
required for      = (Number of cylinder index tracks + 1)
master indexes    / Index entries per track
```

Retrieving and Updating an Indexed Sequential Data Set

Sequential Retrieval and Update

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

- Code DSORG=IS or DSORG=ISU to agree with what you specified when you created the data set, and MACRF=GL, MACRF=SK, or MACRF=PU in the DCB macro.
- 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 whose keys begin with 915. Those records with a date (positions 13 through 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 50 on page 145.

```
//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     DDNAME=INDEXDD,DSORG=IS,MACRF=(GL,SK,PU),      C
          ...     SYNAD=CHECKRR
```

Figure 50. Sequentially Updating an Indexed Sequential Data Set

Direct Retrieval and Update

By using the basic indexed sequential access method (BISAM) to process an indexed sequential data set, you can directly access the records in the data set for:

- 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, if 'S' is specified on the READ macro, the system provides dynamic buffering each time a read request is made. (See Figure 51 on page 148.)

To ensure that the requested record is in virtual storage before you start processing, you must issue a WAIT or CHECK macro. If you issue a WAIT macro, you must test the exception code field of the DECB. If you issue a CHECK macro, the system tests the exception code field in the DECB. If an error analysis routine has not been specified and a CHECK is issued, and an error situation exists, the program is abnormally terminated with a system completion code of XX'01' For both WAIT and CHECK, if you want 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 and if you plan to use the same DECB again to rewrite the updated record using a WRITE K macro, you should not set the field to 0. If you do, your record may not be rewritten properly.

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 or issue a FREEDBUF macro if dynamic buffering was used. If you issue several READ KU or WRITE K macros before checking the first one, you may destroy some of your updated records unless the records are from different blocks.

When you are using 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. ENQ must be issued before the SETL macro, and DEQ must be issued after the ESETL macro. 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: If there is the possibility that your task and another task will be simultaneously accessing 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 unpredictable.

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 and rname operands 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 *Supervisor Services and Macro Instructions*.)

For subtasking, I/O requests should be issued by the task that owns the DCB or a task that will remain active as long as the DCB is open. If the task that issued the I/O request terminates, the storage used by its data areas (such as IOBs) may be freed or queuing switches in the DCB work area may be left set on, causing another task issuing an I/O request to the DCB to program check or to enter the wait state. For example, if a subtask issues and completes a READ KU I/O request, the IOB created by the subtask is attached to the DCB update queue. If that subtask terminates, and subpool zero is not shared with the subtask owning the DCB, the IOB storage area is freed and the integrity of the ISAM update queue is destroyed. A request from another subtask, attempting to use that queue, may cause unpredictable abends. As another example, if a WRITE KEY NEW is in process when the subtask terminates, 'WRITE-KEY-NEW-IN-PROCESS' bit is left set on. If another I/O request is issued to the DCB, the request is queued but cannot proceed.

Direct Update with Exclusive Control—Indexed Sequential Data Set: In the example shown in Figure 51 on page 148, 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 through 10, and the update information is in positions 11 through 30. The update information replaces data in positions 31 through 50 of the indexed sequential data record.

```
//INDEXDD   DD      DSNAME=SLATE.DICT,DCB=(DSORG=IS,BUFNO=1,...),---
//TAPEDD    DD      ---

          ...
ISUPDATE  START   0
          ...
NEXTREC   GET     TPDATA,TPRECORD
          ENQ     (RESOURCE,ELEMENT,E,,SYSTEM)
          READ    DECBRW,KU,,'S',MF=E    Read into dynamically
*                                        obtained buffer
          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     ISUPDATE-ISRECORD      Update record
                  (L'UPDATE,3),UPDATE
          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      ERROR                  If not, go to error
*                                        routine
          FREEDBUF DECBRW,K,ISDATA       Otherwise, free buffer
          MVC     ISKEY,KEY              Key placed in ISRECORD
          MVC     ISUPDATE,UPDATE        Updated information
*                                        placed in ISRECORD
          WRITE   DECBRW,KN,,WKNAREA,'S',MF=E Add record to data
*                                        set
          WAIT    ECB=DECBRW
          TM      DECBRW+24,X'FD'        Test for errors
          BM      ERROR
          DEQ     (RESOURCE,ELEMENT,,SYSTEM)  Release exclusive
*                                       control
          B       NEXTREC
WKNAREA   DS      4F                     BISAM WRITE KN work
*                                         field
ISRECORD  DS      0CL50                  50-byte record from
*                                         ISDATA
          DS      CL19                   DCB First part of
*                                         ISRECORD
```

Figure 51 (Part 1 of 2). Directly Updating an Indexed Sequential Data Set

```
ISKEY       DS      CL10                    Key field of ISRECORD
            DS      CL1                     Part of ISRECORD
ISUPDATE    DS      CL20                    Update area of ISRECORD
*
            ORG     ISUPDATE                Overlay ISUPDATE with
TPRECORD    DS      0CL30                   TPRECORD 30-byte record
KEY         DS      CL10                    from TPDATA DCB Key
*                                            for locating
UPDATE      DS      CL20                    ISDATA record update
RESOURCE    DC      CL8'SLATE'              information or new data
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 51 (Part 2 of 2). Directly Updating an Indexed Sequential Data Set

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

Note the use of the FREEDBUF macro instruction in Figure 51. Usually, the FREEDBUF macro 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 51, because the read operation was unsuccessful, the FREEDBUF macro 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 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 merely 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, because it is more efficient than FREEDBUF. For example, in Figure 51, the FREEDBUF macro could be eliminated, because 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 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. 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. If you use a WRITE KN macro 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.

Do not try to use the DECBLGTH field to determine the length of a record read, because DECBLGTH is for use with writing records, not reading them. If you are reading fixed-length records, the length of the record read is in DCBLRECL, and if you are reading variable-length records, the length is in the record descriptor word (RDW).

Direct Update—Indexed Sequential Data Set with Variable-Length Records: In Figure 52, 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 through 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,MF=E Write transaction record
*                                              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      DCB    DDNAME=INDEXDD,DSORG=IS,MACRF=(RUSC,WUAC),SYNAD=CHECKERR
TPDATA      DCB    ---
            ...
```

Figure 52. Directly Updating an Indexed Sequential Data Set with Variable-Length Records

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.

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 53 on page 153.

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.

For BISAM, if you add a record that has the same key as a record in the data set, a "duplicate record" condition is indicated in the exception code. However, if you specified the delete option and the record in the data set is marked for deletion, the condition is not reported and the new record replaces the existing record. (For more information about exception codes, see *Data Administration: Macro Instruction Reference.*)

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, 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.

When you use the PUT macro (resume load), records are added to the prime data area until the space originally allocated is filled. After 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 under "Creating an Indexed Sequential Data Set" on page 133.

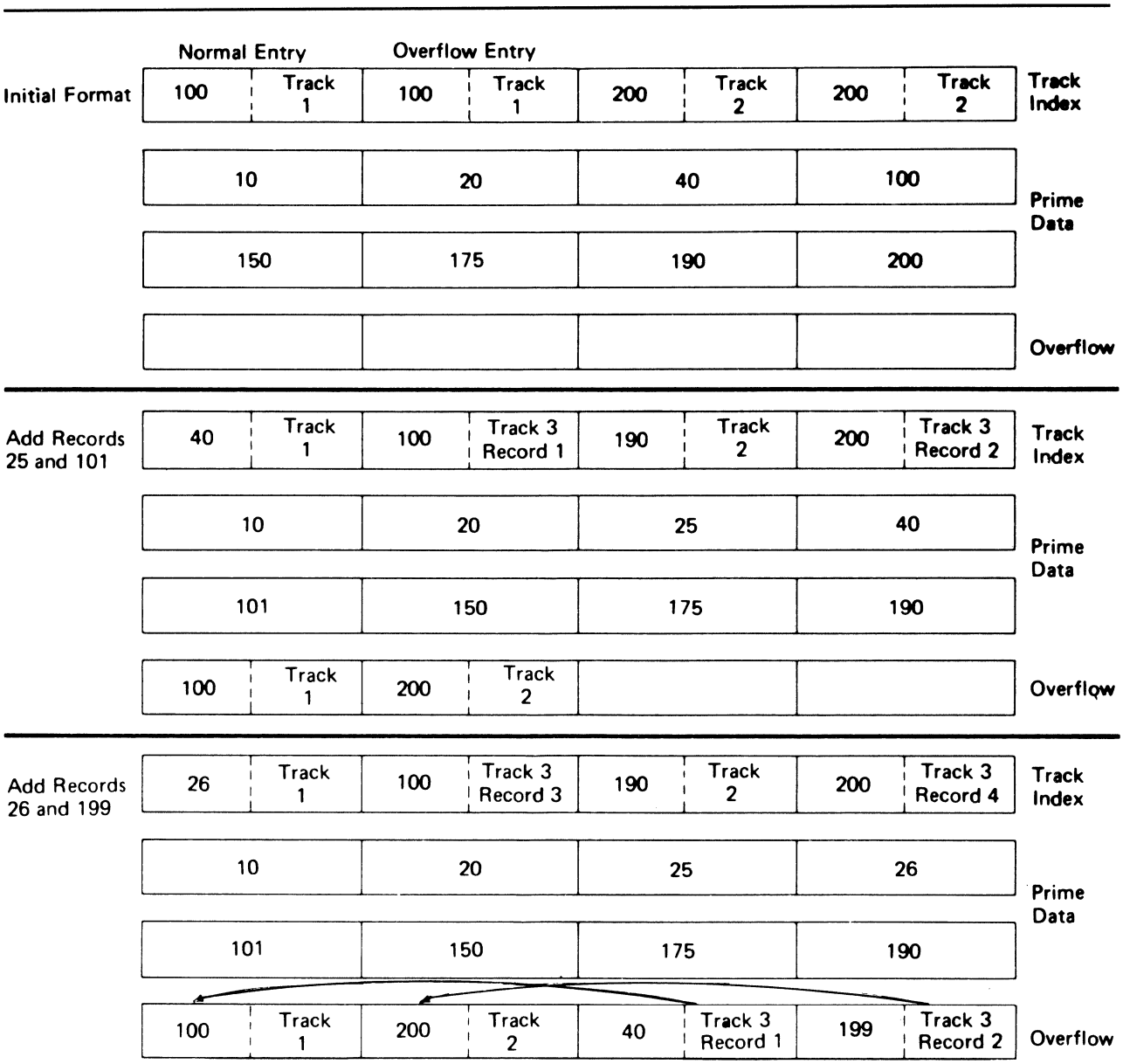


Figure 53. Adding Records to an Indexed Sequential Data Set

In order to add records with successively higher keys using the PUT macro (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 or the EXTEND option is specified in the OPEN macro.
- The data set must have been successfully closed when it was created or when records were previously added using the PUT macro.

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 (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:

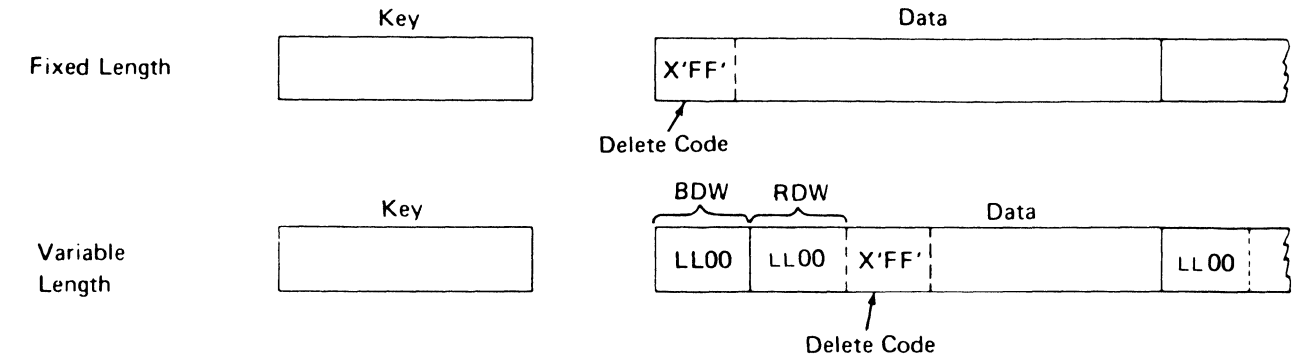
- You can reorganize the data set in two passes by writing it sequentially into another area of direct access storage or magnetic tape and then re-creating it in the original area.
- You can reorganize the data set in one pass by writing it directly into another area of direct access storage. In this 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 54 on page 155, 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 the same as any other records, and you should check them for the delete code.

Note that a WRITE KN (key new) to a data set containing variable-length records removes all the deleted records from that prime data track.

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.



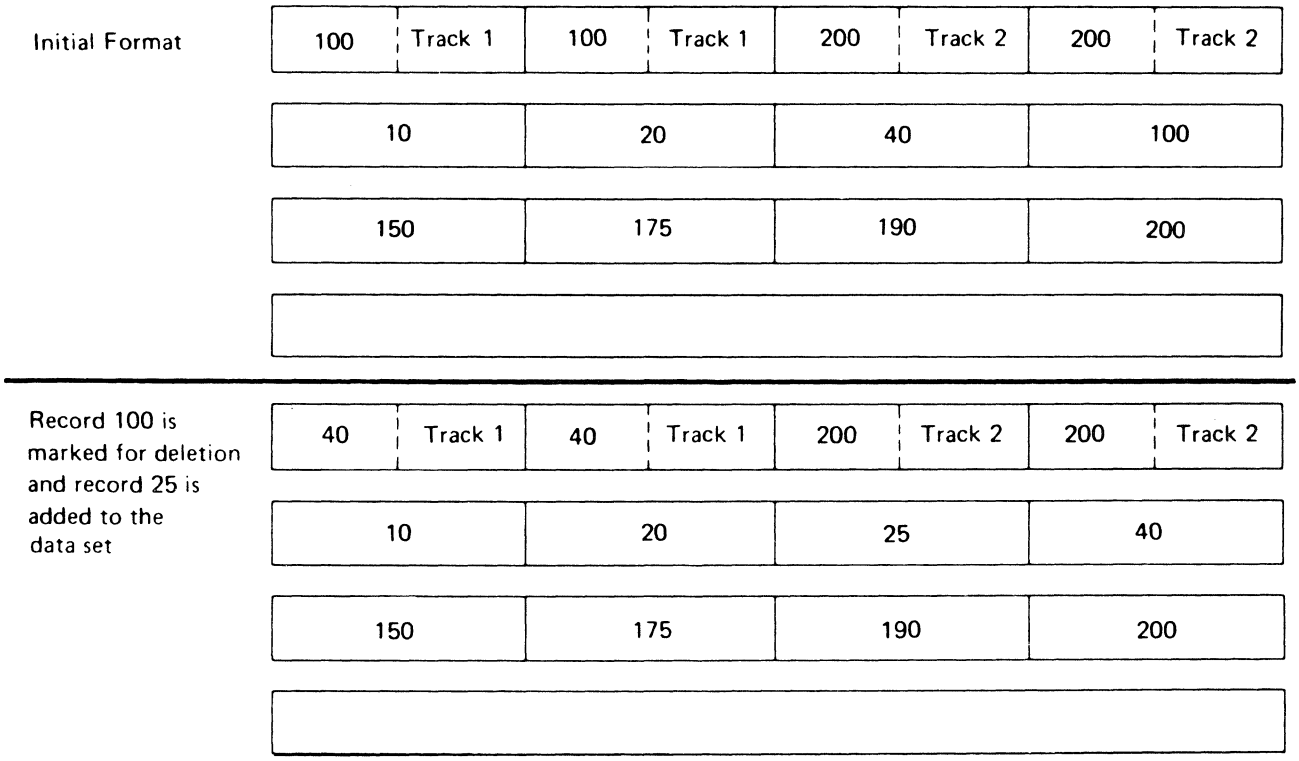
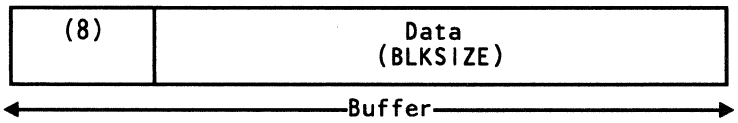


Figure 54. Deleting Records from an Indexed Sequential Data Set

Indexed Sequential Buffer and Work Area Requirements

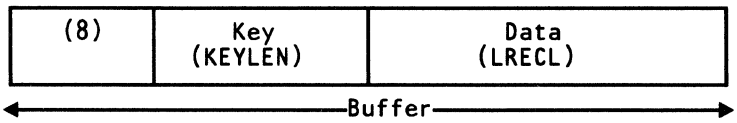
The only case in which you will ever have to compute the buffer length (BUFL) requirements for your program occurs when you use the BUILD or GETPOOL macro to construct the buffer area. If you are creating an indexed sequential data set (using the PUT macro), each buffer must be 8 bytes longer than the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

```
Buffer length = 8 + Block size
```



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:

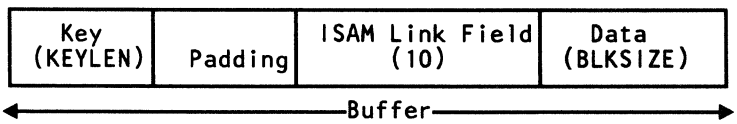
```
Buffer length = 8 + Key length + Record length
```



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

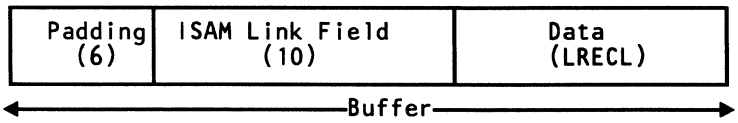
For fixed-length unblocked records when both the key and data are to be read and for variable-length unblocked records, padding is added so that the data will be on a doubleword boundary, that is:

```
Buffer length = Key length + Padding + 10 + Block size
```



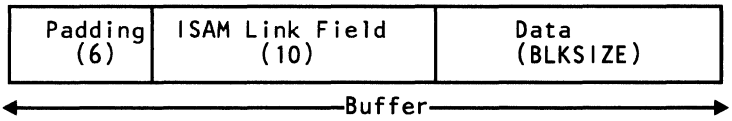
For fixed-length unblocked records when only data is to be read:

```
Buffer length = 16 + LRECL
```



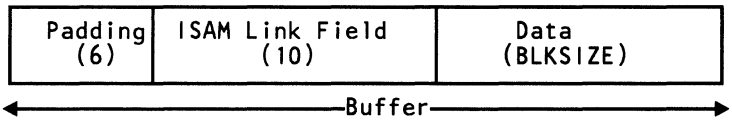
For fixed-length blocked records:

```
Buffer length = 16 + Block size
```



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:

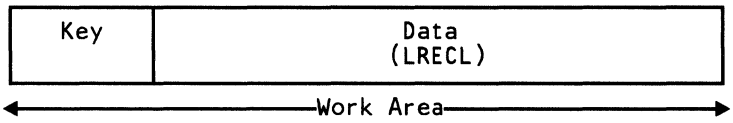
```
Buffer length = 12 or 16 + Block size
```



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.

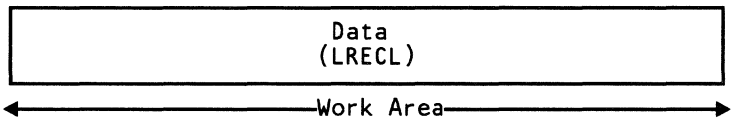
The size of the work area is given by:

```
Work area = Key length + Record length
```



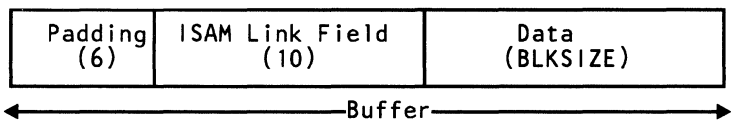
If you are reading only the data portion of fixed-length unblocked records or variable-length records, the work area is the same size as the record, that is:

```
Work area = Record length
```



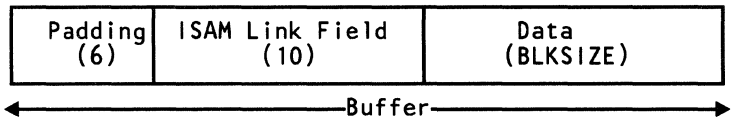
When you use the basic access technique to update records in an indexed sequential data set, the key length field need not be considered in determining your buffer requirements. The area for fixed-length records must be:

```
Buffer length = 16 + Block size
```



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 save processing time by adding fixed-length or variable-length records to a data set by using the MSWA parameter of the DCB macro 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 and data set information from the DSCB obtained with the OBTAIN macro. (The DEVTYPE and OBTAIN macros are discussed in *System-Data Administration*.)

Note that you can use the DEVTYPE macro 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. The maximum value for SMSW is 65535.

For calculating the size of the work area, see the storage device capacities shown in Figure 58 on page 189 and the device overhead formulas given in "Estimating Space Requirements" on page 188.

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

```
SMSW = (DS2HIRPR) (BLKSIZE + 8) + LRECL + KEYLEN
```

The formula for fixed-length unblocked records is

```
SMSW = (DS2HIRPR) (KEYLEN + LRECL + 8) + 2
```

The value for DS2HIRPR is in the index (format-2) DSCB. *Debugging Handbook* 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 storage than the second method.

The first method is as follows:

```
SMSW = DS2HIRPR (BLKSIZE + 8) + LRECL + KEYLEN + 10
```

The second method is as follows:

```
SMSW = ( (Trk Cap - Bn + 1) / Block length) (BLKSIZE)
       + 8 (DS2HIRPR) + LRECL + KEYLEN
       + 10 + (REM - N - KEYLEN)
```

In all the above formulas, the terms BLKSIZE, LRECL, KEYLEN, and SMSW are the same as the parameters in the DCB macro (Trk Cap=track capacity). REM is the remainder of the division operation in the formula and N is the first constant in the block length formulas described in Figure 59 on page 190. (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 virtual 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 65535. If you do not use this technique, the index on the volume must be searched. If the high-level index is greater than 65535 bytes in length, your request for the high-level index in storage is ignored.

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 to gain access to the DCBNCRHI field (see Chapter 5, "Specifying a Data Control Block and Initializing Data Sets" on page 39). 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 = (Number of Tracks in High-Level Index)
       (Number of Entries per Track)
       (Key Length + 10)
```

The formula for calculating the number of tracks in the high-level index is in "Calculating Space Requirements for an Indexed Sequential Data Set" on page 139. 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 want 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 to terminate sequential processing. You can then begin processing at another point in the data set. If you do not specify a SETL macro prior to retrieving the data, the system assumes default SETL values. (See the GET and SETL macros in *Data Administration: Macro Instruction Reference.*)

SETL—Specify Start of Sequential Retrieval

The SETL macro 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 class (key prefix), or an actual address of a prime data record.

The key class concept is useful because you do not have to know the whole key of the first record to be processed. A key class comprises all the keys that begin with identical characters. The key class is defined by specifying the desired characters of the key class at the address specified in the lower-limit operand of the SETL macro and setting the remaining characters to the right of the key class to binary zeros.

To use actual addresses, you must keep a record 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.

Normally, when a data set is created with the delete option specified, deleted records cannot be retrieved using the QISAM retrieval mode. When the delete option is not specified in the DCB, the SETL macro options function as follows:

SETL B	Start at the first record in the data set.
SETL K	Start with the record having the specified key.
SETL KH	Start with the record whose key is equal to or higher than the specified key.
SETL KC	Start with the first record having a key that falls into the specified key class.
SETL I	Start with the record found at the specified direct access address in the prime area of the data set.

Because the DCBOPTCD field in the DCB can be changed after the data set is created (by respecifying the OPTCD in the DCB or DD card), it is possible to retrieve deleted records. In this case, SETL functions as noted above.

When the delete option is specified in the DCB, the SETL macro options function as follows:

SETL B	Start retrieval at the first undeleted record in the data set.

SETL K	Start retrieval at the record matching the specified key, if that record is not deleted. If the record is deleted, an NRF (no record found) indication is set in the DCBEXCD field of the DCB, and SYNAD is given control.
SETL KH	Start with the first undeleted record whose key is equal to or higher than the specified key.
SETL KC	Start with the first undeleted record having a key that falls into the specified key class or follows the specified key class.
SETL I	Start with the first undeleted record following the specified direct access address.

With the delete option not specified, QISAM retrieves and handles records marked for deletion as nondeleted records.

Note: Regardless of the SETL or delete option specified, the NRF condition will be posted in the DCBEXCD field of the DCB, and SYNAD is given control if the key or key class:

- Is higher than any key or key class in the data set
- Does not have a matching key or key class in the data set

ESETL—End Sequential Retrieval

The ESETL macro 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 must be executed before another SETL macro (described above) using the same DCB is executed.

Note: If the previous SETL macro completed with an error, an ESETL macro should be executed before another SETL macro.

Chapter 13. Generation Data Groups

A generation data group is a group of related cataloged data sets. The way these data sets are cataloged is what makes them a generation data group. Within a generation data group, the generations can have like or unlike DCB attributes and data set organizations. If the attributes and organizations of all generations in a group are identical, the generations can be retrieved together as a single data set. Each data set within a generation data group is called a generation data set. Generation data sets are sometimes called generations.

There are advantages to grouping related data sets. Because the catalog management routines can refer to the information in a special index—called a generation index—in the catalog:

- All of the data sets in the group can be referred to by a common name.
- The operating system is able to keep the generations in chronological order.
- Outdated or obsolete generations can be automatically deleted by the operating system.

The management of a generation data group depends upon the fact that generation data sets have sequentially ordered names—absolute and relative names—that represent their age. The absolute generation name is the representation used by the catalog management routines in the catalog. Older data sets have smaller absolute numbers. The relative name is a signed integer used to refer to the latest (0), the next to the latest (-1), and so forth, generation. The relative number can also be used to catalog a new generation (+1).

A generation data group base is created in an integrated catalog facility or VSAM catalog before the generation data sets are cataloged. A generation data group is represented in the integrated catalog facility or VSAM catalog by a generation data group base entry. The access method services DEFINE command is used to create the generation data group base. See *Access Method Services Reference* for information on how to define and/or catalog generation data sets in an integrated catalog facility or VSAM catalog. See *Utilities* for information on how to define and/or catalog generation data sets in an OS CVOL.

Absolute Generation and Version Numbers

An absolute generation and version number is used to identify a specific generation of a generation data group. The generation and version numbers are in the form *GxxxxVyy*, where *xxxx* is an unsigned 4-digit decimal generation number (0001 through 9999) and *yy* is an unsigned 2-digit decimal version number (00 through 99). For example:

- A.B.C.G0001V00 is generation data set 1, version 0, in generation data group A.B.C.
- A.B.C.G0009V01 is generation data set 9, version 1, in generation data group A.B.C.

The number of generations and versions is limited by the number of digits in the absolute generation name, that is, 9999 for generations and 100 for versions.

The generation number is automatically maintained by the system. The number of generations kept depends on the size of the generation index. For example, if the size of the generation index allows ten entries, the ten latest generations may be maintained in the generation data group.

The version number allows you to perform normal data set operations without disrupting the management of the generation data group. For example, if you want to update the second generation in a 3-generation group, replace generation 2, version 0, with generation 2, version 1. Only one version is kept for each generation.

A generation can be cataloged using either absolute or relative numbers. When a generation is cataloged, a generation and version number is placed as a low level entry in the generation data group. In order to catalog a version number other than V00, you must use an absolute generation and version number.

A new version of a specific generation can be cataloged automatically by specifying the old generation number along with a new version number. For example, if generation A.B.C.G0005V00 is cataloged and you now create and catalog A.B.C.G0005V01, the new entry is cataloged in the location previously occupied by A.B.C.G0005V00. This process removes the old entry from the catalog but does not scratch the old version. To scratch the old version and make its space available for reallocation, a DD card, describing the data set to be deleted, with DISP=(OLD,DELETE) should be included at the time the data set is to be replaced by the new version.

Relative Generation Number

As an alternative to using absolute generation and version numbers when cataloging or referring to a generation, you can use a relative generation number. To specify a relative number, use the generation data group name followed by a negative integer, a positive integer, or a 0, enclosed in parentheses. For example, A.B.C(-1). A.B.C(+1), or A.B.C(0).

The value of the specified integer tells the operating system what generation number to assign to a new generation, or it tells the system the location of an entry representing a previously cataloged generation.

When you use a relative generation number to catalog a generation, the operating system assigns an absolute generation number and a version number of V00 to represent that generation. The absolute generation number assigned depends on the number last assigned and the value of the relative generation number that you are now specifying. For example if, in a previous job generation, A.B.C.G0005V00 was the last generation cataloged, and you specify A.B.C(+1), the generation now cataloged is assigned the number G0006V00. Though any positive relative generation number can be used, a number greater than 1 may cause absolute generation numbers to be skipped. For example, if you have a single step job, and the generation being cataloged is a +2, one generation number is skipped. However, in a multiple step job, one step may have a +1 and a second step a +2, and no numbers are skipped in this case.

Note: If you do not specify a volume in the JCL for a new generation data set, and the data set is not opened, that data set is not cataloged.

Programming Considerations for Multiple Step Jobs

One of the reasons for using generation data groups is to allow the system to maintain a given number of related cataloged data sets. If you attempt to delete or uncatalog any but the oldest of the data sets of a generation data group in a multiple step job, catalog management can lose orientation within the data group. This can cause the deletion, uncataloging, or retrieval of the wrong data set when referring to a specified generation. The rule is, if you delete a generation data set in a multiple step job, do not refer to any older generation in subsequent job steps.

Also, it is recommended that, in a multiple step job, you catalog or uncatalog data sets using JCL instead of IEHPROGM or a user program. Because ALLOCATION/UNALLOCATION monitors data sets during job execution and is not aware of the functions performed by these programs, data set orientation may be lost or conflicting functions may be performed in subsequent job steps.

When you use a relative generation number to refer to a generation that was cataloged in a previous job, the relative number has the following meaning:

- A.B.C(0) refers to the latest existing cataloged entry.
- A.B.C(-1) refers to the next-to-the-latest entry, and so forth.

When cataloging is requested via JCL, all actual cataloging occurs at step termination, but the relative generation number remains the same throughout the job. Because this is so:

- A relative number used in the JCL refers to the same generation throughout a job.
- A job step that terminates abnormally may be deferred for a later step restart. If the step cataloged a generation data set via JCL, you must change all relative

generation numbers in the succeeding steps via JCL before resubmitting the job.

For example, if the succeeding steps contained the relative generation numbers:

- A.B.C(+1), that refers to the entry cataloged in the terminated job step, or
- A.B.C(0), that refers to the next to the latest entry, or
- A.B.C(-1), that refers to the latest entry, prior to A.B.C(0),

you must change them as follows before the step can be restarted: A.B.C(0), A.B.C(-1), A.B.C(-2), and so forth.

Generation Data Group Naming for ISO/ANSI/FIPS Version 3 Labels

In a Version 3 ISO/ANSI/FIPS label (LABEL=(,AL)), the generation number and version number are maintained separately from the file identifier. During label processing, the generation number and version number are removed from the generation data set name. The generation number is placed in the generation number field (file label 1 positions 36 through 39), and the version number is placed in its position on the same label (position 40 and 41). The file identifier portion of a Version 3 HDR1/EOF1/EOV1 label contains the generation data set name without the generation number and version number.

For Version 3 labels, you must observe the following specifications created by the generation data group naming convention.

- Data set names whose last 9 characters are of the form *.GnnnnVnn* (n is 0 through 9) can only be used to specify GDG data sets. When a name ending in *.GnnnnVnn* is encountered, it is automatically processed as a GDG. The generation number *Gnnnn* and the version number *Vnn* are separated from the rest of the data set name and placed in the generation number and version number fields.
- Tape data set names for GDG files are expanded from a maximum of 8 user-specified characters to 17 user-specified characters. (The tape label file identifier field has space for 9 additional user-specified characters because the generation number and version number are no longer contained in this field.)
- A generation number of all zeros is not valid, and will be treated as an error during label validation. The error appears as a "RANG" error in message IEC512I (IECIEUNK) during the label validation installation exit.
- In an MVS system-created GDG name, the version number will always be 0. (MVS will not increase the version number by 1 for subsequent versions.) To obtain a version number other than 0, you must explicitly specify the version number (for example, A.B.C.G0004V03) when the data set is created. You must also explicitly specify the version number to retrieve a GDG with a version number other than 0.
- Because the generation number and version number are not contained on the identifier of HDR1, generations of the same GDG will have the same name.

Therefore, an attempt to place more than one generation of a GDG on the same volume will result in an ISO/ANSI/FIPS standards violation in a system supporting Version 3, and MVS will enter the validation installation exit.

Creating a New Generation

To create a new generation data set, you must first allocate space for the generation, then catalog the generation.

Allocating a Generation

To take full advantage of the facilities of the system, the allocation can be patterned after a previously allocated generation in the same group. This is accomplished by the specification of DCB attributes for the new generation as described below.

If you are using absolute generation and version numbers, DCB attributes for a generation can be supplied directly in the DCB parameter of the DD statement defining the generation to be created and cataloged.

If you are using relative generation numbers to catalog generations, DCB attributes can be supplied either: (1) by creating a model DSCB on the volume on which the index resides (the volume containing the catalog) or (2) by referring to a cataloged data set for the use of its attributes. Attributes can be supplied before you catalog a generation, when you catalog it, or at both times, as follows:

1. Create a model DSCB on the volume on which your index resides. You can provide initial DCB attributes when you create your model; however, you need not provide any attributes at this time. Because only the attributes in the data set label are used, the model data set should be allocated with SPACE=(TRK,0) to conserve direct access space. Initial or overriding attributes can be supplied when you create and catalog a generation.[1] To create a model DSCB, include the following DD statement in the job step that builds the index or in any other job step that precedes the step where you create and catalog your generation.

```
//name DD DSNAME=datagrpname,DISP=(,KEEP),SPACE=(TRK,(0)),
//        UNIT=yyyy,VOLUME=SER=xxxxxx,
//        DCB=(applicable subparameters)
```

   The DSNAME is the common name by which each generation is identified; xxxxxx is the serial number of the volume containing the catalog. If no DCB subparameters are wanted initially, you need not code the DCB parameter.

---

1 Only one model DSCB is necessary for any number of generations. If you plan to use only one model, do not supply DCB attributes when you create the model. When you subsequently create and catalog a generation, include necessary DCB attributes in the DD statement referring to the generation. In this manner, any number of generation data groups can refer to the same model. Note that the catalog and model data set label are always located on a direct access volume, even for a magnetic tape generation data group.

2. You do not need to create a model DSCB if you can refer to a cataloged data set whose attributes are identical to those you desire or to an existing model DSCB for which you can supply overriding attributes. To refer to a cataloged data set for the use of its attributes, specify DCB=(dsname) on the DD statement that creates and catalogs your generation. To refer to an existing model, specify DCB=(modeldscbname, your attributes) on the DD statement that creates and catalogs your generation.

Passing a Generation

A new generation may be passed when created. That generation may then be cataloged in a succeeding job step or deleted at the end of the job as in normal disposition processing when DISP=(,PASS) is specified on the DD statement.

However, after a generation has been created with DISP=(NEW,PASS) specified on the DD statement, another new generation for that data group must not be cataloged until the passed version has been deleted or cataloged. To do so would cause the wrong generation to be used when referencing the passed generation data set. If that data set was later cataloged, a bad generation would be cataloged and a good one lost.

For example, if A.B.C(+1) was created with DISP=(NEW,PASS) specified on the DD statement, then A.B.C.(+2) must not be created with DISP=(NEW,CATLG) until A.B.C(+1) has been cataloged or deleted.

By using the proper JCL, the advantages to this support are:

- JCL will not have to be changed in order to rerun the job.
- The lowest generation version will not be deleted from the index until a valid version is cataloged.

Creating an ISAM Data Set as Part of a Generation Data Group

To create an indexed-sequential data set as part of a generation data group, you must: (1) create the indexed-sequential data set separately from the generation group and (2) use IEHPROGM to put the indexed-sequential data set into the generation group.

In an integrated catalog facility and VSAM catalogs, use access method services commands to catalog the data set. In an OS CVOL, use the RENAME function to rename the data set. Then use the CATLG function to catalog the data set. For instance, if MASTER is the name of the generation data group, and *GggggVvv* is the absolute generation name, you would code the following:

```
RENAME DSNAME=ISAM,VOL=3380=SCRTCH,NEWNAME=MASTER.GggggVvv
CATLG DSNAME=MASTER.GggggVvv,VOL=3380=SCRTCH
```

Retrieving a Generation

A generation may be retrieved through the use of job control language procedures. Any operation that can be applied to a nongeneration data set can be applied to a generation. For example, a generation can be updated and reentered in the catalog, or it can be copied, printed, punched, or used in the creation of new generation or nongeneration data sets.

You can retrieve a generation by using either relative generation numbers or absolute generation and version numbers.

Because two or more jobs can compete for the same resource, generation data groups should be updated with care, as follows:

- No two jobs running concurrently should refer to the same generation data group. As a partial safeguard against this situation, use absolute generation and version numbers when cataloging or retrieving a generation in a multiprogramming environment. If you use relative numbers, a job running concurrently may update the generation data group index, perhaps cataloging a new generation which you will then retrieve in place of the one you wanted.

- Even when using absolute generation and version numbers, a job running concurrently might catalog a new version of a generation or perhaps delete the generation you wanted to retrieve. For this reason, some degree of control should be maintained over the execution of job steps referring to generation data groups.

Building a Generation Data Group Index

A generation data group is managed via the information found in a generation index. (Note that an alias name cannot be assigned to the highest level of a generation index.) The BLDG function of IEHPROGM builds the index. The BLDG function also indicates how older or obsolete generations are to be handled when the index is full. For example, when the index is full, you may want to empty it, scratch existing generations, and begin cataloging a new series of generations.

After the index is built, a generation can be cataloged by its generation data group name and either an absolute generation and version number or a relative generation number.

Examples of how to build a generation data group index are found in *Utilities*, under IEHPROGM.

Chapter 14. I/O Device Control Macros

The operating system provides you with several macros for controlling input/output devices. Each is, to varying degrees, device dependent. Therefore, you must exercise care if you want 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.

CNTRL—Control an I/O Device

The CNTRL macro 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 cannot be used with an input data set containing variable-length records on the card reader.

If you specify OPTCD=H in the DCB parameter field of the DD statement, you can use the CNTRL macro to position DOS tapes that contain embedded DOS checkpoint records. The CNTRL macro 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 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—Printer Setup

The SETPRT macro instruction is used to control how information is printed. It is used with the 3800 Printing Subsystem and with various other universal character set (UCS) printers.

For the IBM 3800 Printing Subsystem, the SETPRT macro instruction is used to initially set or dynamically change the printer control information. For additional information on how to use the SETPRT macro with the 3800 printer, see *IBM 3800 Printing Subsystem Programmer's Guide.*

For printers that have a universal character set (UCS) buffer and optionally, a forms control buffer (FCB), the SETPRT macro instruction is used to specify the UCS and/or FCB images to be used. Note that universal character sets for the various printers are not compatible. The three formats of FCB images (the FCB image for the 3800 Printing Subsystem, the 4248 format FCB and the 3211 format FCB) are incompatible. The 3211 format FCB is used by the 3203, 3211, 4248, 3262 Model 5, and 4245 printers.

IBM-supplied UCS images, UCS image tables, FCB images, and character arrangement table modules are included in the SYS1.IMAGELIB at system generation time. For 1403, 3203, 3211, 3262 Model 5, 4245, and 4248 printers, user-defined character sets can be added to SYS1.IMAGELIB. For a description of how images are added to SYS1.IMAGELIB and how band names/aliases are added to image tables, see *System-Data Administration.* For the 3800, user-defined character arrangement table modules, FCB modules, GRAPHIC modules, copy modification modules, and library character sets can be added to SYS1.IMAGELIB as described in *Utilities.* For information on building a 4248 format FCB (which can also be used for the IBM 3262 Model 5 printer), see *Utilities.*

The FCB contents can be selected from the system library (or an alternate library if you are using a 3800), or defined in your program through the exit list of the DCB macro instruction, as discussed under -- Heading id 'exlst' unknown --.

For a non-3800 printer, the specified UCS or FCB image should be found in one of the following:

- SYS1.IMAGELIB
- Image table (UCS Image only)
- DCB exit list for an FCB

If the image is not found, the operator is asked to specify an alternate image name or cancel the request.

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

BSP—Backspace a Magnetic Tape or Direct Access Volume

The BSP macro 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 if the track overflow option was specified or if the CNTRL, NOTE, or POINT macro instruction is used. The BSP macro should be used only when other device control macros could not be used for backspacing.

Any attempt to backspace across a file mark will result in a return code of X'04' in register 15 and your tape or direct access volume will be positioned after the file mark. This means you cannot issue a successful backspace command after your EODAD routine is entered unless you first reposition the tape or direct access volume into your data set. (CLOSE TYPE=T can position you at the end of your data set.)

You can use the BSP macro 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 for DOS 7-track tapes written in translate mode.

NOTE—Return the Relative Address of a Block

The NOTE macro 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 data set on the volume currently being processed.

For magnetic tape, the address is in the form of a 4-byte relative block address. If TYPE=REL is specified or defaults, the address provided by the operating system is returned in register 1. If TYPE=ABS is specified, the physical block identifier of a data block on tape is returned in register 0. The relative block address or the block identifier can later be used as a search argument for the POINT macro.

For a direct access device, the address is in the form of a 4-byte relative track address. The address provided by the operating system is returned in register 1, and 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 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. For disk, if a write operation follows the POINT macro, all of the track following the write operation is erased, unless the data set is opened for UPDAT. POINT is not meant to be used before a WRITE macro when a data set is opened for UPDAT. If you specify OPTCD=H in the DCB parameter field of the DD statement, you can use the POINT macro to position DOS tapes that contain embedded checkpoint records. The POINT macro cannot be used to backspace DOS 7-track tapes that are written in data convert mode and contain embedded checkpoint records.

If you specify TYPE=ABS, you can use the physical block identifier as a search argument to locate a data block on tape. The identifier may be provided from the output of a prior execution of the NOTE macro.

When using the POINT macro for a direct access device that is opened for OUTPUT, OUTIN, or INOUT, and the record format is not standard, the number of blocks per track may vary slightly.

SYNCDEV—Control Data Synchronization

The SYNCDEV macro controls data synchronization for devices supporting buffered write mode. Data still in the buffer may not yet reside on the final recording medium. This is referred to as data that is not synchronized. You can either

- Request information regarding synchronization.
- Demand that synchronization occur based on a specified number of data blocks that are allowed to be buffered. If zero is specified, synchronization will always occur.

When SYNCDEV completes successfully (return code 0), a value will be returned that indicates the number of data blocks remaining in the control unit buffer.

Chapter 15. Protecting Data

Control of confidential data in a data set is provided through password protection. 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 supplying a predefined password.

Password Protection for Non-VSAM Data Sets

Password protection as described here applies to non-VSAM data sets only. For information on password protection for VSAM data sets, see *Access Method Services Reference*.

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, or for a TSO job by the TSO user.

- 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 when a job is being requested by the open or EOV routine, the job is terminated by the system. For a SCRATCH or RENAME request, a return code is given.

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 *Magnetic Tape Labels and File Structure* or in the identifier data set control block (DSCB) as shown in *Debugging Handbook*. After you have requested security protection for magnetic tapes, you cannot remove it with JCL unless you re-create the data set and scratch the protected data set.

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 named PASSWORD, which must be created on the system-residence volume. 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 when you enter its password in the PASSWORD data set. You can use the PROTECT macro 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. *System-Data Administration* describes how to maintain the PASSWORD data set, including the PROTECT macro instruction; *Utilities* describes the IEHPROGM utility program.

RACF Protection for Non-VSAM Data Sets

Resource Access Control Facility (RACF) protection as described here applies to non-VSAM data sets, tape data sets, and tape volumes. For information on RACF protection for VSAM data sets, see *VSAM Administration Guide*. For detailed information on RACF protection for data sets, see *RACF General Information Manual* and *RACF Security Administrator's Guide*.

RACF is an IBM licensed program that provides access control by identifying and verifying users and authorizing access to DASD and tape data sets and volumes. A generic profile can protect both DASD data sets and tape data sets. You may use RACF to provide access control to tape volumes that have no labels (NL), standard labels (SL), ISO/ANSI/FIPS labels (AL), or to tape volumes that are referenced with bypass label processing (BLP).

You may define a data set to RACF automatically or explicitly. The automatic definition occurs when space is allocated for the DASD data set, if the user has the automatic data set protection attribute or if PROTECT=YES is coded on the DD statement. The explicit definition of a data set to RACF is by use of the RACF command language.

RACF protection of tape data sets is provided on a volume basis or on a data set basis. A tape volume is defined to RACF explicitly by use of the RACF command language or automatically. A tape data set is defined to RACF whenever a data set is opened for OUTPUT, OUTIN, or OUTINX and RACF tape data set protection is active, whether or not the data set is the first file in a sequence. All data sets on a tape volume are RACF protected if the volume is RACF protected.

Five levels of access authority are possible in a RACF-defined data set or tape volume.

ALTER
: You have total control over the data set. If you define the data set or tape volume to RACF, you have ALTER access authority. With ALTER authority, you can read and write the data set or tape volume, rename the data set, and scratch the data set, and you may authorize other users access to the tape volume or data set.

CONTROL
: For non-VSAM data sets, CONTROL authority is equivalent to UPDATE authority.

UPDATE
You are authorized to open the data set or tape volume for OUTPUT and all other open options.

READ
You are authorized to open the data set or tape volume for INPUT only.

NONE
You are not authorized to open the data set or tape volume.

If a data set is defined to RACF and is password protected, access to the data set is authorized only through RACF authorization checking. If a tape volume is defined to RACF and the data set(s) on the tape volume is password protected, access to any of the data sets is authorized only through RACF authorization checking of the volume. Data set password protection is bypassed.

To protect multivolume non-VSAM DASD and tape data sets, you must define each volume of the data set to RACF as part of the same volume set. When a RACF-protected data set is opened for output and extended to a new volume, the new volume is automatically defined to RACF as part of the same volume set. When a multivolume physical-sequential data set is opened for output and any of the data set's volumes are defined to RACF, either each subsequent volume must be RACF protected as part of the same volume set, or the data set must not yet exist on the volume. When a RACF protected multivolume tape data set is opened for output, either each subsequent volume must be RACF protected as part of the same volume set, or the tape volume must not yet be defined to RACF. If the first volume opened is not RACF protected, no subsequent volume may be RACF protected. If a multivolume data set is opened for input (or a nonphysical-sequential data set is opened for output), no such consistency check is performed when subsequent volumes are accessed.

ISO/ANSI/FIPS Version 3 installation exits that execute under RACF will receive control during ANSI volume label processing. Control will go to the RACHECK preprocessing and postprocessing installation exits. The same IECIEPRM exit parameter list passed to ANSI installation exits will be passed to the RACF installation exits if the accessibility code is any alphabetic character from A through Z. For more information, see *Data Facility Product: Customization*.

Erasing RACF Protected DASD Data Sets

You can create or alter RACF profiles to include an ERASE option for DASD data sets. DFP tests for this option, and, if you have specified ERASE, it overwrites the DASD space with zeros before making it available for reallocation.

If you have specified ERASE, the entire data set area is overwritten when you use any of the following:

- The DELETE subparameter in the JCL DISP parameter on a DD statement
- The TSO DELETE command (for non-VSAM objects)
- The SCRATCH macro instruction
- The SCRATCH control statement for the IEHPROGM utility program

If the data set is sequential or partitioned and you have specified ERASE, the released area is overwritten when you use any of the following:

- The RLSE subparameter in the JCL SPACE parameter on a DD statement
- The PARTREL macro instruction

See the *RACF General Information* manual and associated publications for more information about specifying and using the ERASE option.

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 55). In the case of direct access volumes, the data set label is the data set control block (DSCB).

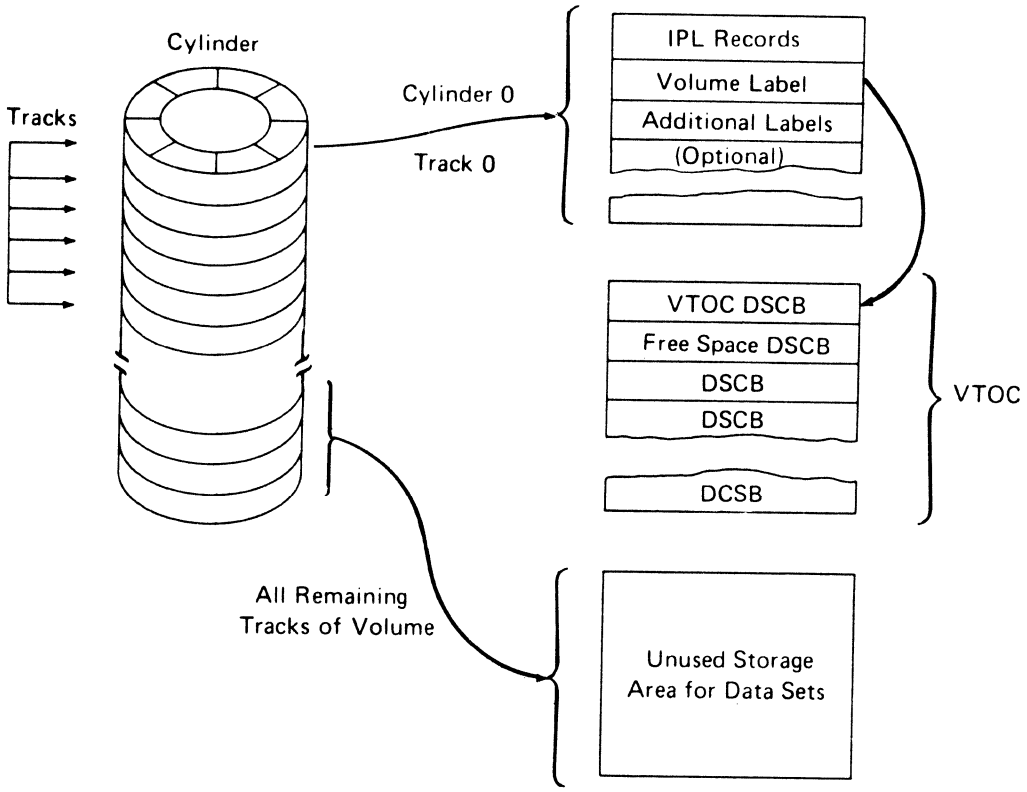


Figure 55. Direct Access Labeling

Volume-Label Group

The volume-label group immediately follows the first two initial program loading (IPL) records on track 0 of cylinder 0 of the volume. It consists of the initial volume label at record 3 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 can be processed by an installation routine that is incorporated into the system.

The format of the direct access volume label group is shown in Figure 56.

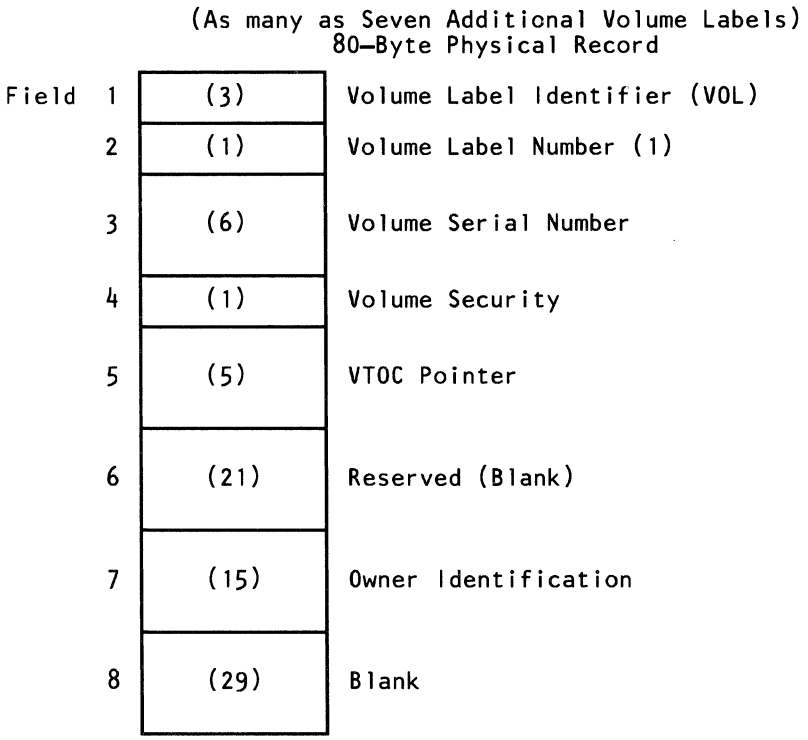


Figure 56. Initial Volume Label

Initial Volume Label Format

The 80-byte initial volume label is preceded by a 4-byte key containing VOL1.

Volume Label Identifier (VOL): Field 1 identifies a 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 X'F1'.

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 through 999999), but may be any 1 to 6 alphameric or national (#, $, @) characters, or a hyphen (X'60'). If this field is less than 6 characters, it is padded on the right with blanks.

Volume Security: Field 4 is reserved for use by installations that want to provide security for volumes. Make this field a X'C0' unless you have your own security processing routines.

VTOC Pointer: Field 5 of direct access volume label 1 contains the address of the VTOC in the form of CCHHR.

Reserved: Field 6 is reserved for possible future use. Leave it blank.

Owner Name and Address Code: Field 7 contains a unique identification of the owner of the volume.

All the bytes in Field 8 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, in addition to space allocation and other control information, contains operating system data, device-dependent information, and data set characteristics. There are seven kinds of DSCBs, each with a different purpose and a different format number. For an explanation of Format-1 through Format-6 DSCBs, see *System—Data Administration*. Format 0 DSCBs are used to indicate empty space in the VTOC.

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 57.

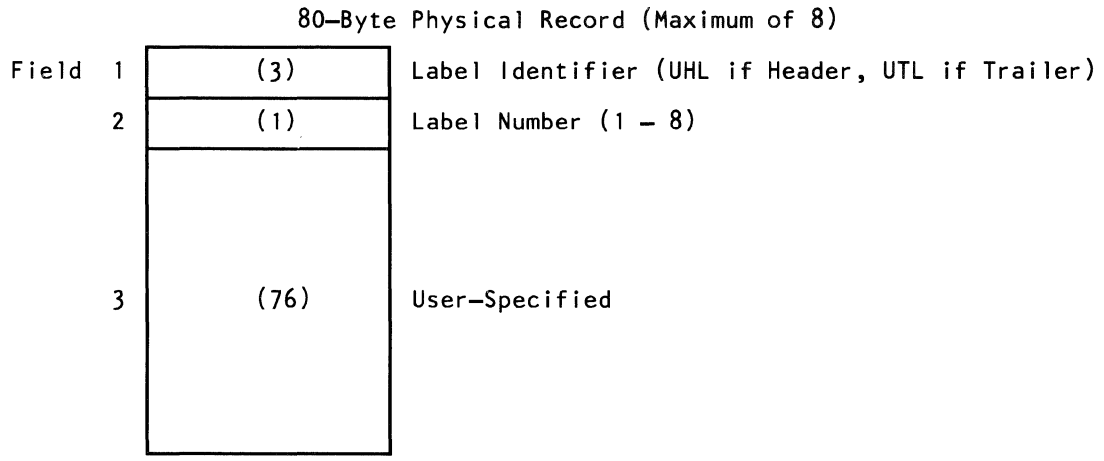


Figure 57. User Header and Trailer Labels

Each group can include as many as 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.

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 to 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.

The machine code control characters for a printer are:

Print—Then Act	Action	Act Immediately without Printing
X'01'	Print only (no space)	
X'09'	Space 1 line	X'0B'
X'11'	Space 2 lines	X'13'
X'19'	Space 3 lines	X'1B'
X'89'	Skip to channel 1	X'8B'
X'91'	Skip to channel 2	X'93'

Print—Then Act	Action	Act Immediately without Printing
X'99'	Skip to channel 3	X'9B'
X'A1'	Skip to channel 4	X'A3'
X'A9'	Skip to channel 5	X'AB'
X'B1'	Skip to channel 6	X'B3'
X'B9'	Skip to channel 7	X'BB'
X'C1'	Skip to channel 8	X'C3'
X'C9'	Skip to channel 9	X'CB'
X'D1'	Skip to channel 10	X'D3'
X'D9'	Skip to channel 11	X'DB'
X'E1'	Skip to channel 12	X'E3'

The machine code control characters for a card read punch device are as follows:

Control Code	Action
X'01'	Select stacker 1
X'41'	Select stacker 2
X'5A'[1]	Change from line mode to page mode
X'81'	Select stacker 3

[1]The 3800 Model 3 all-point-addressable mode uses this code to change from comparability to page mode.

Other 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 (ANSI). Whenever IBM publications refer to ANSI control characters, they are coded 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
Code	**Action after Punching a Card**
V	Select punch pocket 1
W	Select punch pocket 2
X'5A'[1]	Change from line to page mode

[1]The 3800 Model 3 all-point-addressable mode uses this code

These control characters include those defined by ANSI FORTRAN. If any other character is specified, it is interpreted as 'b' or a V, depending on whether it is for a printer or a punch; no error indication is returned.

Appendix C. Allocating Space on Direct Access Volumes

When direct access storage space is required for a data set, you can specify the amount of space needed, the device type, and, optionally, the volume. The operating system selects the device and allocates the space accordingly.

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 how it is to be allocated.

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 or block allocation unless your device supports Define Extent or Locate Record.

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 = primary quantity (number of blocks)
      100 = secondary quantity, to be allocated if the primary
            quantity is not enough (in blocks)
```

From this information, the operating system estimates and allocates the number of tracks required. Space is always in complete 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 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: The amount of space required can be expressed in tracks or cylinders, as in these examples:

```
//  DD SPACE=(TRK,(100,5)), . . .
//  DD SPACE=(CYL,(3,1)), . . .
```

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 about the number of tracks and the beginning address, as in this example:

```
//  DD SPACE=(ABSTR,(500,15)),UNIT=3380, . . .
```

where 500 tracks are required, beginning at relative track 15, which is cylinder 1, track 0.

Allocation of Mass Storage System (MSS) Virtual Volumes: When the data set is to be stored on an MSS virtual volume, a volume group (MSVGP) parameter may be specified instead of using the SPACE parameter on the DD card. Before the MSVGP parameter can be used, the volume group must be identified to MSS by the utility program IDCAMS.

Allocation of MSS virtual volume space should be in multiples of cylinders, with secondary allocation a multiple of the primary to ensure maximum space usage and minimum fragmentation.

Additional Space Allocation Options: The DD statement provides you with much flexibility in specifying space requirements. The options are described in detail in *JCL*.

Estimating Space Requirements

To determine how much space your data set requires, you must consider these variables for the device type:

- Track capacity
- Tracks per cylinder
- Cylinders per volume
- Data length (block size)
- Key length
- Device overhead

Figure 58 on page 189 lists the physical characteristics of several direct access storage devices.

Device	Volume Type	Maximum Block size per Track[1]	Tracks per Cylinder	Number of Cylinders[2]	Total Capacity[1,2]
2305-2	Disk	14660	8	96	11258880
3330/3333[3] (Model 1)	Disk	13030	19	404	100018280
3330/3333 (Model 11)	Disk	13030	19	808	200036560
3340/3344[4]	Disk	8368	12	696 (70 megabytes) 348 (35 megabytes)	69889536 34944768
3350	Disk	19069	30	555	317498850
3375	Disk	32760[5]	12	959	409868928
3380	Disk	32760[5]	15	885	630243900
3380 (Models AD4 and BD4)	Disk	32760[5]	15	885	630243900
3380 (Models AE4 and BE4)	Disk	32760[5]	15	1770	1260487800

1 Capacity indicated in bytes (when R0 is used by the IBM programming system).

2 Excluding alternate cylinders.

3 The Mass Storage System (MSS) virtual volumes assume the characteristics of the 3330/3333, Model 1.

4 The 3344 is functionally equivalent to the 3340 Model 70.

5 The largest record that can be written on a track is 35616 bytes for the 3375 and 47476 bytes for all 3380 models. However, for these devices, the largest block size supported by the standard access methods is 32760 bytes.

Figure 58. Direct Access Storage Device Capacities

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, and so forth. 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 59 on page 190. Note that any fraction of a byte must be ignored in any operation of the

calculation. For example, if the formula gives 15.644 bytes, you must allocate 15 bytes.

Note: You may choose to use the TRKCALC macro to perform track capacity calculations. See *System—Data Administration* for further information on TRKCALC.

Bytes Required by Each Data Block[1]

Device	Track Capacity	Blocks with Keys	Blocks without Keys
2305-2	14858[2]	289+KL+DL	198+DL
3330/3333[3] (Model 1 or 11)	13165	191+KL+DL	135+DL
3340/3344	8535[2]	242+KL+DL	167+DL
3350	19254[2]	267+KL+DL	185+DL
3375	36000	224+((KL+191)/32)(32)+ ((DL+191)/32)(32)	224+((DL+191)/32)(32)

Figure 59. Direct Access Device Overhead Formulas

Legend:
DL is data length.
KL is key length.

Notes:

1 Use modulo-32 arithmetic when calculating key length and data length terms in your equations. Compute these terms first, then round up to the nearest increment of 32 bytes before completing the equation.

2 This value is different from the maximum block size per track because the formula for the last block on the track includes an overhead for this device.

3 The Mass Storage System (MSS) virtual volumes assumes the characteristics of the 3330/3333, Model 1.

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, 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.

Appendix D. ISO/ANSI/FIPS Record Control Word and Segment Control Word

Translation of ISO/ANSI/FIPS Record Control Word

The ISO/ANSI/FIPS record control word (RCW) is expressed in ISCII/ASCII characters and is 4 bytes long (see Figure 60). Note that the RCW is different from the code in the IBM record descriptor word (RDW). The RDW, expressed in binary, is the internal data management equivalent of the ISO/ANSI/FIPS RCW.

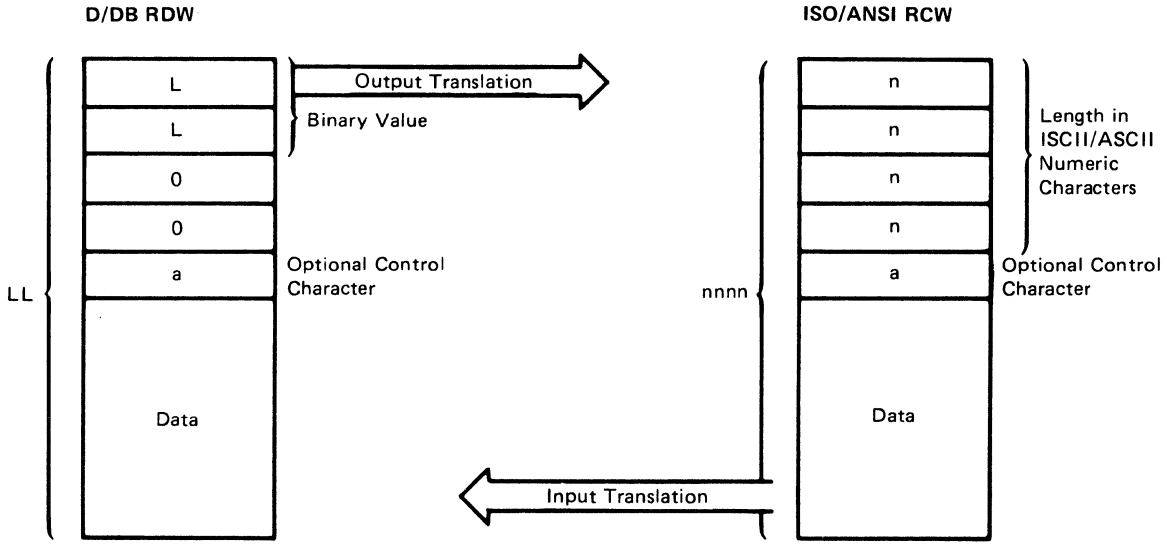
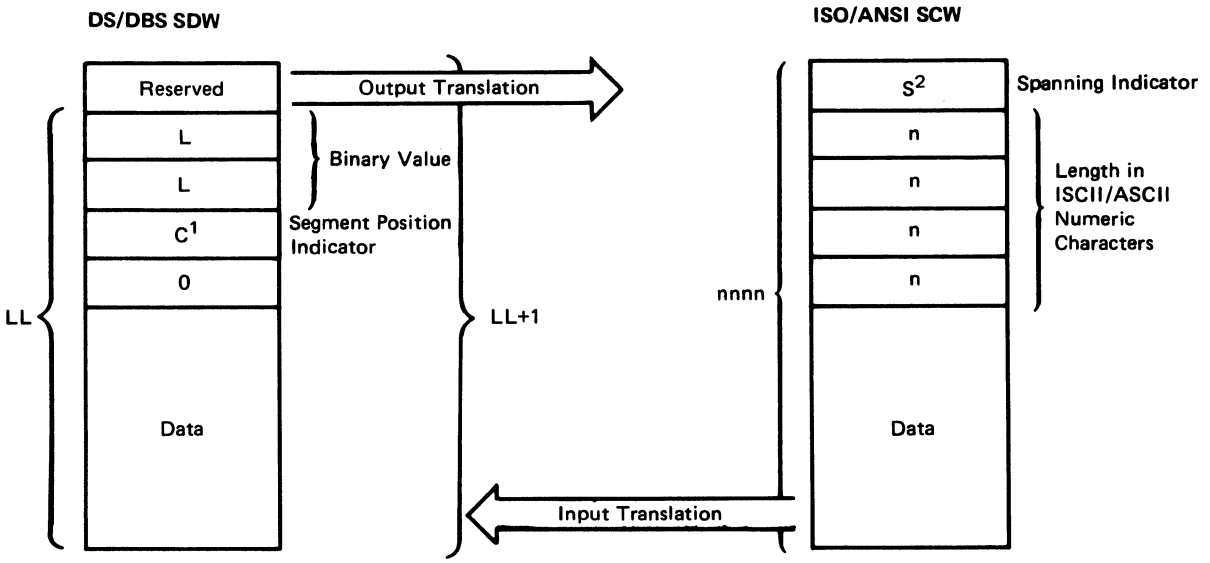


Figure 60. Translation of ISO/ANSI/FIPS Record Control Word to D/DB Record Descriptor Word

Translation of ISO/ANSI/FIPS Segment Control Word

The ISO/ANSI/FIPS segment control word (SCW) is expressed in ISCII/ASCII characters and is 5 bytes in length. (See Figure 61.) Note that the SCW is different from the code in the IBM segment descriptor word (SDW). The SDW is the internal data management equivalent of the ISO/ANSI/FIPS SCW. Only 4 bytes are used by data management, but the user buffer area must accommodate an extra byte to allow for translation from the ISO/ANSI/FIPS SCW. The SDW is expressed in binary.



[1] C values for SDW (2 low order bits)

00 = only segment of record
01 = first segment of record
11 = intermediate segment of record
10 = last segment of record

[2] S values for SCW (ASCII characters)

0 = only segment of record
1 = first segment of record
2 = intermediate segment of record
3 = last segment of record

Figure 61. Translation of ISO/ANSI/FIPS Segment Control Word to DS/DBS Segment Descriptor Word

Glossary of Terms and Abbreviations

The following terms are defined as they are used in this book. If you do not find the term you are looking for, see the index or the *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699.

A. ANSI control code (value of RECFM)

ABE. abnormal end (value of EROPT)

ABEND. abnormal end (macro instruction)

ABSTR. absolute track (value of SPACE)

ACC. accept erroneous block (value of EROPT)

AFF. affinity (channel separation parameter of DD statement or unit affinity value of UNIT)

AL. American National Standard Labels

ANSI. American National Standards Institute

ASCII. American National Standard Code for Information Interchange

AUL. American National Standard user labels (value of LABEL)

B. blocked records (value of RECFM)

BCDIC. binary coded decimal interchange code

BDAM. basic direct access method

BDW. block descriptor word

BFALN. buffer alignment (operand of DCB)

BFTEK. buffer technique (operand of DCB)

BISAM. basic indexed sequential access method

BLDL. build list (macro instruction)

BLKSIZE. blocksize (operand of DCB)

BPAM. basic partitioned access method

BPI. bytes per inch

BSAM. basic sequential access method

BSM. backspace past tapemark and forward space over tapemark (operand of CNTRL)

BSP. backspace one block (macro instruction)

BSR. backspace over a specified number of blocks (records) (operand of CNTRL)

BUFCB. buffer pool control block (operand of DCB)

BUFL. buffer length (operand of DCB)

BUFNO. buffer number (operand of DCB)

BUFOFF. buffer offset (length of ASCII block prefix by which the buffer is offset; operand of DCB)

CCW. channel command word

CONTIG. contiguous space allocation (value of SPACE)

CNTRL. control (macro instruction)

CSW. channel status word

CYLOFL. number of tracks for cylinder overflow records (operand of DCB)

D. format-D (ISCII/ASCII variable-length) records (value of RECFM)

DA. direct access (value of DEVD or DSORG)

DAU. direct access unmovable data set (value of DSORG)

DB. ISCII/ASCII variable-length, blocked records (value of RECFM)

DBS. ISCII/ASCII variable-length, blocked spanned records (value of RECFM)

DCB. data control block (control block name or macro instruction or parameter on DD statement)

DCBD. data control block dummy section (macro instruction)

DD. data definition(statement)

DEB. data extent block

DECB. data event control block

DEN. magnetic tape density (operand of DCB)

DEVD. device-dependent (operand of DCB)

DISP. data set disposition (parameter of DD statement)

DS. ISCII/ASCII variable-length, spanned records (value of RECFM)

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)

FIPS. Federal Information Processing Standard

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)

GCR. group coded recording(tape recording mode)

GL. GET macro, locate mode (value of MACRF)

GM. GET macro, move mode (value of MACRF)

H. DOS tapes with embedded checkpoint records (parameter of OPTCD)

HA. home address

ICF catalog. integrated catalog facility catalog

INOUT. input then output (operand of OPEN)

I/O. input/output

IOB. input/output block

IPL. initial program load

IRG. interrecord gap

IS. indexed sequential (value of DSORG)

ISAM. indexed sequential access method

ISCII. International Standard Code for Information Interchange

ISO. International Organization for Standardization

ISU. indexed sequential unmovable (value of DSORG)

JCL. job control language

JFCB. job file control block

JFCBE. job file control block extension for 3800 printer

K. 1024

KEYLEN. key length (operand of DCB)

LPA. link pack area

LPALIB. link pack area library

LRECL. logical record length (operand of DCB)

LRI. logical record interface

M. machine control code (value of RECFM)

MACRF. macro instruction form (operand of DCB)

MOD. modify data set (value of DISP)

MSHI. main storage for highest-level index (operand of DCB)

MSS. Mass Storage System

MSVC. Mass Storage Volume Control

MSWA. main storage for work area (operand of DCB)

NCP. number of channel programs (operand of DCB)

NOPWREAD. no password required to read a data set (value of LABEL)

NRZI. nonreturn-to-zero-inverted (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)

OPTCD. optional services code (operand of DCB)

OS CVOL. operating system control volume

OS/VS. operating system/virtual storage

OUTIN. output then input (operand of OPEN)

PCI. program-controlled interruption

PDAB. parallel data access block

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 method

QSAM. queued sequential access method

RACF. Resource Access Control Facility

RDBACK. read backward (operand of OPEN)

RDW. record descriptor word

RECFM. record format (operand of DCB)

RKP. relative key position (operand of DCB)

RLSE. release unused space (DD statement)

RPS. rotational position sensing

S. standard format records (value of RECFM)

SDW. segment descriptor word

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)

SVC. supervisor call

SVCLIB. supervisor call library

SYNAD. synchronous error routine address (operand of DCB)

SYSIN. system input stream

SYSOUT. system output stream

T. track overflow option (value of RECFM); user-totaling(value of OPTCD)

TIOT. task I/O table

TRC. table reference character

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, blocked records (value of RECFM)

VBS. variable-length, blocked, spanned records (value of RECFM)

VS. variable-length, spanned records

VSAM catalog. virtual storage access method catalog

VTOC. volume table of contents

XLRI. extended logical record interface

Index

C

D

E

G

H

I

J

K

L

M

N

O

P

Q

R

S

W

X

Numerics

Reader's Comment Form

MVS/XA Data Administration Guide
GC26-4140-2

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you wish a reply, give your name, company, mailing address, and telephone number.

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL _____

Previous TNL _____

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Thank you for your cooperation.

GC26-4140-2

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM®

MVS/XA Data Administration Guide (File No. S370-34) Printed in U.S.A. GC26-4140-2

Reader's Comment Form

MVS/XA Data Administration Guide
GC26-4140-2

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you wish a reply, give your name, company, mailing address, and telephone number.

_____

_____

_____

_____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL _____

Previous TNL _____

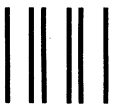
Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Thank you for your cooperation.

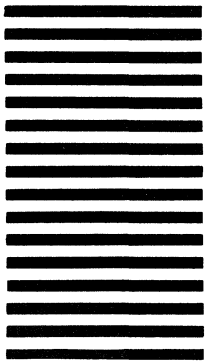
GC26-4140-2

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

MVS/XA Data Administration Guide (File No. S370-34) Printed in U.S.A. GC26-4140-2

MVS/Extended Architecture
Data Administration Guide

File Number S370-34

GC26-4140-02

Printed in U.S.A.