File Number S360-30
Order Number GC26-3746-0

OS

IBM Systems Reference Library

IBM System/360 Operating System
Data Management Services

The material in this manual was formerly in Section II of IBM System/360 Operating System Supervisor and Data Management Services, Order Number GC28-6646. That manual is now titled IBM System/360 Operating System Supervisor Services.

This manual describes the input/output facilities of the IBM System/360 Operating System. It contains information on record formats, data set organization, access methods, direct access device characteristics, data set disposition, and space allocation.

Intended mainly for the assembler-language programmer, this book is a guide to using the macro instructions described in IBM System/360 Operating System Supervisor and Data Management Macro Instructions, Order Number GC28-6647. This book does not discuss macro instructions used for graphics, teleprocessing, optical readers, optical reader-sorters, or magnetic character readers. These macro instructions are discussed in separate publications that are listed in the IBM System/360 Bibliography, Order Number GA22-6822.

This book contains information about the time sharing option (TSO), which is not yet available. This information should be used only for planning purposes until the time sharing option becomes available.

First Edition (January 1971)

This publication corresponds to Release 20 of the IBM System/360 Operating System. It consists of what used to be Section II of the manual IBM System/360 Operating System Supervisor and Data Management Services, Order Number GC28-6646. The changes to the text and to the illustrations are indicated by a vertical line in the margin to the left of the change.

Changes have been made to the text because data management now translates data from American Standard Code for Information Interchange (ASCII) format on magnetic tape to EBCDIC format within the operating system and back and because of the addition of the time sharing option (TSO) to the MVT version of the IBM System/360 Operating System. There are other technical changes throughout the book.

Information in this book changes from time to time. Before using this manual with IBM systems, consult the latest IBM 360 SRL Newsletter, Order Number GN20-0360, for the editions that are current and applicable.

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

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

Preface

Part 1 of this publication, "Introduction to Data Management," contains general information about the services that the operating system provides for input and output. It describes how data sets are named, how they can be found, and how they are stored.

Part 2, "Data Management Processing Techniques," is a guide to using the data management macro instructions. It discusses general programming techniques -- queued v. basic access techniques, error handling, buffering -- and then treats each data organization technique separately. There are sections on sequential organization (BSAM and QSAM), partitioned organization (BPAM), indexed sequential organization (ISAM), and direct organization (BDAM).

Part 3, "Data Set Disposition and Space Allocation," contains the information you need to decide how much space to allocate for a data set. It also discusses what the operating system does with your data set when your job is done, and how you can tell it what to do.

The appendixes describe direct access labels and ANSI and machine control characters.

This book assumes you have a basic knowledge of the operating system and of System/360 assembler language. Two books that contain information about these subjects are:

IBM System/360 Operating System:

Concepts and Facilities, Order Number GC28-6335

Assembler Language, Order Number GC28-6514

In the examples in this book, the macro instructions are coded in just enough detail to make the examples clear. For a complete description of all the operands and options available with any of the macro instructions discussed here, see

IBM System/360 Operating System Supervisor and Data Management Macro Instructions, Order Number GC28-6647.

This book also assumes that if you are using the MVT version of the control program with the time sharing option (TSO), you understand how to use TSO. Specifically, the book assumes that you are familiar with the concepts discussed in the following books:

IBM System/360 Operating System:

Time Sharing Option Command Language, Order Number GC28-6732, which describes the TSO command language that a terminal user must use to request computing services.

Time Sharing Option Guide, Order Number GC28-6698, which describes the concepts, features, and capabilities of TSO.

Time Sharing Option Programmer's Guide, Order Number GC28-6764, which relates TSO processing to batch processing and describes data handling by TSO, extending the command language, and writing conversational programs.

If you are using the operating system without TSO, ignore the sections "Time Sharing Option (TSO) Terminal Access" and "Sequential Data Sets -- Terminal Control" and the TSO information in the discussions of these macro instructions:

GET	TRUNC
PUTX	RELSE
PUT	CNTRL
WRITE	PRTOV
READ	BSP
CHECK	SETPRT
FEOV	NOTE

Other publications referred to in the text are:

IBM System/360 Operating System:

Job Control Language Reference, Order Number GC28-6704

Job Control Language User's Guide, Order Number GC28-6703

Sort/Merge, Order Number GC28-6662

Supervisor Services, Order Number GC28-6646

System Control Blocks, Order Number GC28-6628

System Generation, Order Number GC28-6554

System Programmer's Guide, Order Number GC28-6550

Tape Labels, Order Number GC28-6680

Contents

Figures

Examples

Part 1: Introduction to Data Management

Data Set Characteristics

The manner in which data is transferred between main storage and external devices is of paramount importance in most data processing applications. The data management function of the System/360 Operating System assists you in achieving maximum efficiency in managing the mass of data associated with the many programs that are processed at an installation. To attain this objective, data management facilities have been designed to provide systematic and effective means of organizing, identifying, storing, cataloging, and retrieving all data, including loadable programs, processed by the operating system.

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

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

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

Data from a direct access volume, a remote terminal, or a tape, and data organized sequentially or as in a library, may all be requested by you in essentially the same way. In addition, data management provides:

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

Control of confidential data is provided by the data set security facility of the System/360 Operating System. Using this facility, you can prevent unauthorized access to payroll data, sales forecast data, and all other data sets requiring special security attention. The security-protected data set is available for processing only when a correct password is furnished.

The data access facilities provided by the operating system are a major extension of previous input/output control systems. Input/output routines are provided to efficiently schedule and control the transfer

of data between main storage and input/output devices. Routines are available to:

- Read data.
- Write data.
- Translate data from ASCII (American National Standard Code for Information Interchange) to EBCDIC and back.
- Block and deblock records.
- Overlap reading, writing, and processing operations.
- Read and verify volume and data set labels.
- Write data set labels.
- Automatically position and reposition volumes.
- Detect error conditions and correct when possible.
- Provide exits to user-written error and label routines.

Corresponding to the range of system facilities available for control of data is an equal range of facilities for access to the data. The variety of techniques for gaining access to a data set is derived from two variables: data set organization and data access technique.

Operating System/360 data sets can be organized in four ways:

- Sequential: This is the familiar tape-like structure, in which records are placed in physical rather than logical sequence. Thus, given one record, the location of the next record is determined by its physical position in the data set. The sequential organization is used for all magnetic tapes, and may be selected for direct access devices. Punched tape, punched cards, and printed output are considered to be sequentially organized.
- Indexed Sequential: Records are arranged in collating sequence, according to a key that is a part of every record, on the tracks of a direct access volume. In addition, a separate index or set of indexes maintained by the system gives the location of certain principal records. This permits direct as well as sequential access to any record.
- Direct: This organization is available for data sets on direct access volumes. The records within the data set 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. Records are stored and retrieved directly with addressing specified by you.
- Partitioned: This structure has characteristics of both the sequential and the indexed sequential organizations. Independent groups of sequentially organized data, called members, are in direct access storage. Each member has a simple name stored in a directory that is part of the data set and contains the location of the member's starting point. Partitioned data sets are generally used to store programs. As a result, they are often referred to as libraries.

Requests for input/output operations on data sets through macro instructions are divided into two categories or techniques: the technique for queued access and the technique for basic access. Each

technique is identified according to its treatment of buffering and input/output synchronization 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 the macro instruction capabilities. Also, you may choose a data organization according to the access techniques and processing capabilities available. The code generated by the macro instructions for both techniques is optionally reenterable depending on the form in which parameters are expressed.

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

An important feature of data management is that much of the detailed information needed to store and retrieve data, such as device type, buffer processing technique, and format of output records need not be supplied until the job is ready to be executed. This device independence permits changes to be made in those details without requiring changes in the program. Therefore, you may design and test a program without knowing the exact input/output devices that will be used when it is executed.

Device independence is a feature of both access techniques when you are processing a sequential data set. The degree of device independence achieved is to some extent determined by you. Many useful device-dependent features are available as part of special macro instructions, and achieving device independence requires some selectivity in their use.

DATA SET IDENTIFICATION

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

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

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

Each simple name consists of one to eight 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 qualification levels is possible for a data set name.

To permit different data sets to be processed 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 (e.g., unit type and volume serial number) is specified in a job control statement called the data definition (DD) statement. To gain access to the data set during processing, a reference is made to a data control block associated with the name of the DD statement. Space for a data control block is reserved by a DCB macro instruction when your program is assembled.

DATA SET STORAGE

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

- A reel of magnetic tape.
- A disk pack.
- A bin in a data cell.
- A drum.
- That part of an IBM 2302 disk storage device served by one access mechanism (the device would have either two or four volumes in all).

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

Various groups of labels are used in secondary storage of the System/360 Operating System to identify magnetic tape and direct access volumes, as well as the data sets they contain. Magnetic tape volumes can have standard or nonstandard labels, or they can be unlabeled. Direct access volumes must use standard labels. Standard label support includes a volume label, a data set label for each data set, and optional user labels.

Keeping track of the volume on which a particular data set resides can be a burden and often a source of error. To alleviate this problem, the system provides for automatic cataloging of data sets. A cataloged data set can be retrieved by the system if given only the name of the data set. If the name is qualified, each qualifier corresponds to one of the indexes in the catalog. For example, the data set DEPT58.SMITH.DATA3 is found by searching a master index to determine the location of the index name DEPT58. That index is then searched to find the location of the index SMITH. Finally, that index is searched for DATA3 to find the identification of the volume containing the required 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 (i.e., their generation) can be identified, and are called generation data groups. For example, a data set name LAB.PAYROLL(0) refers to the most recent data set of the group; LAB.PAYROLL(-1) refers to the second most recent data set, etc. The same collection of data set names can be used

repeatedly -- with no requirement to keep track of the volume serial numbers used.

DIRECT ACCESS VOLUMES

Direct access volumes play a major role in the System/360 and System/370 Operating Systems. They are used to store executable programs, including the operating system itself. Direct access storage is also used for data and for temporary working storage. One direct access storage volume may be used for many different data sets, and space on it may be reallocated and reused. A volume table of contents (VTOC) is used to account for each data set and available space on the volume.

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

The volume table of contents describes the contents of the direct access volume. It is a data set that is composed of a series of data set control blocks (DSCB), each of which is composed of one or more control blocks. The VTOC can contain the following data set control blocks:

- A DSCB for each data set on the volume.
- A DSCB that indicates the space allocated to the VTOC itself.
- A DSCB for all tracks on the volume that are available for allocation.

The DSCB for each data set contains the name, description, and location of the data set on the volume. Its size depends on the organization and the number of noncontiguous areas of the data set.

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

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

MAGNETIC TAPE VOLUMES

Because of the sequential organization of magnetic tape devices, the operating system does not require space allocation facilities comparable to those for direct access devices. When a new data set is to be placed on a magnetic tape volume, you must specify the data set sequence number if it is not the first data set on the reel. A volume with standard labels, American National Standards Institute (ANSI) labels, or no labels will be positioned by the operating system so that the data set can be read or written. If the data set has nonstandard labels, the installation must provide volume-positioning in its 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 will indicate the data set sequence number from IPL to IPL and yy will indicate the volume sequence number for the data set. If you specify volume serial numbers for unlabeled volumes on which a data set is to be stored, the system assigns volume serial numbers to any additional volumes required. If data sets residing on unlabeled volumes are to be cataloged or passed, you should specify the volume serial numbers for the volumes required. This will prevent data sets residing on different volumes from being cataloged or passed under identical volume serial numbers. Retrieval of such data sets could result in unpredictable errors.

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

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

For more information on labels for magnetic tape volumes, refer to the Tape Labels manual.

The data on magnetic tape volumes can be either in EBCDIC or in ASCII (American National Standard Code for Information Interchange). ASCII is a seven-bit code consisting of 128 characters. It permits data on magnetic tape to be transferred from one computer to another even though the two computers may be products of different manufacturers.

Data management support of ASCII and of American National Standards Institute tape labels means that data management can translate records on input tapes in ASCII format into EBCDIC format for internal processing and translate the EBCDIC back into ASCII for output. Such input tapes may be sorted into ASCII collating sequence, as explained in the Sort/Merge book.

DATA SET RECORD FORMATS

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

The process of grouping a number of records before writing them on a volume is referred to as blocking. A block is considered to be made up of the data between interrecord gaps (IRG). Each block can consist of one or more records. Blocking conserves storage space on the volume because it reduces the number of interrecord gaps in the data set. In

many cases, blocking also increases processing efficiency by reducing the number of input/output operations required to process a data set.

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

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

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

FIXED-LENGTH RECORDS

The size of fixed-length (format F) records, shown in Figure 13, is constant for all records in the data set. The number of records within a block is usually 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.

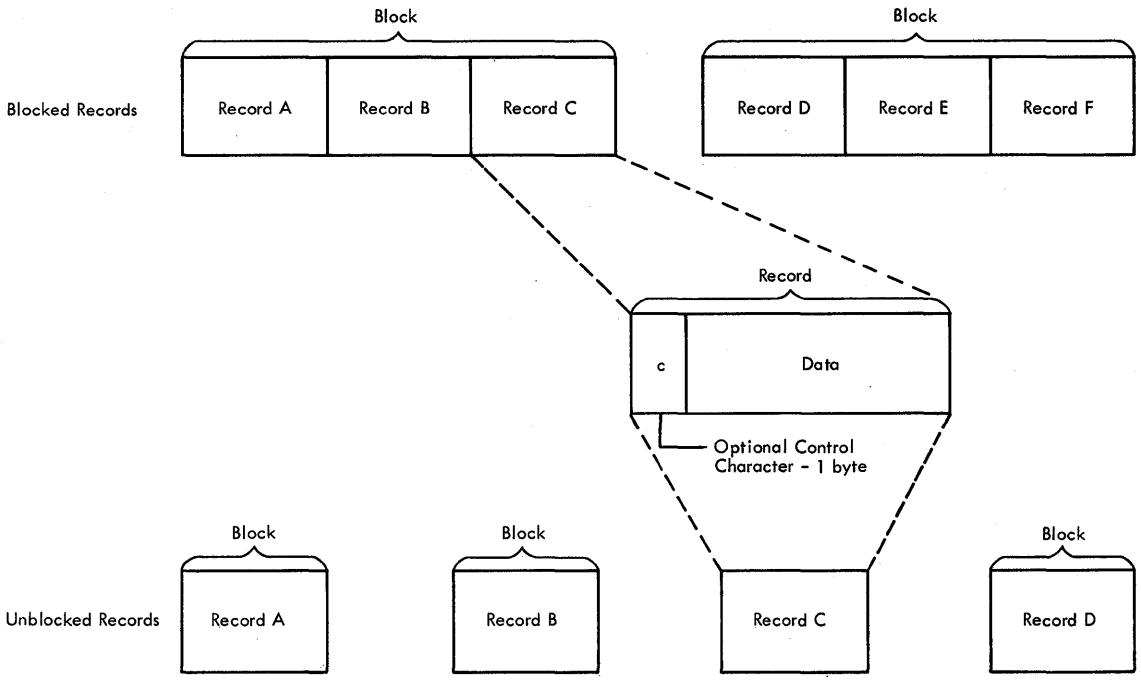


Figure 1. Fixed-Length Records

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

A sequential data set is said to contain records in standard format F if:

- All records in the data set are format F.
- Every track except the last is filled to capacity (no room for another record).
- No blocks except the last are truncated.

Standard format F data sets can be read from direct access storage more efficiently than data sets with truncated blocks because the system can determine the location of each block to be read. If you use standard format F records to create a sequential data set in direct access storage, the system puts the same number of blocks on each track.

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

For ASCII tapes, format F records are the same as described above, with two exceptions:

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

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

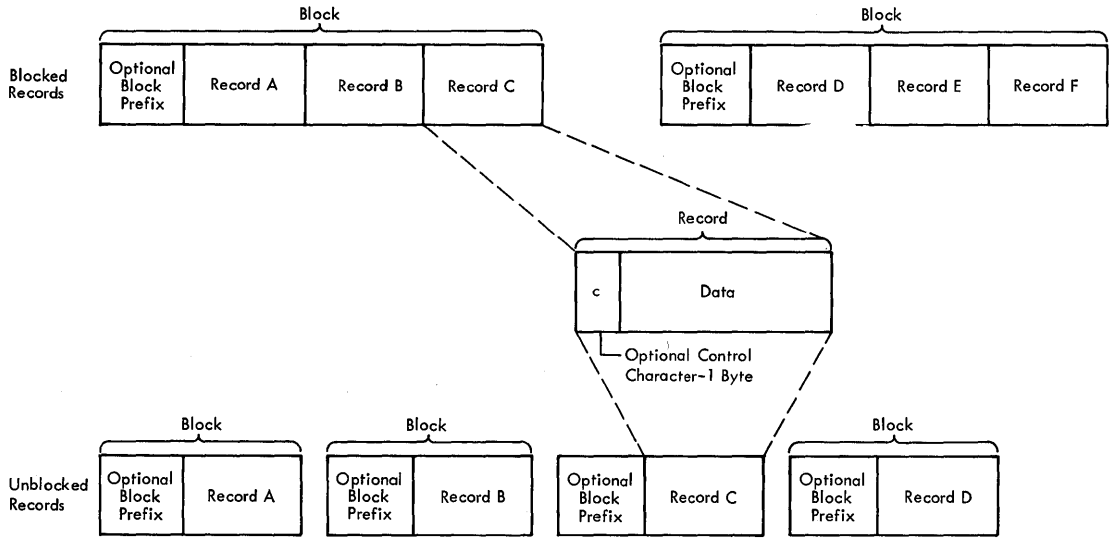


Figure 2. Fixed-Length Records for ASCII Tapes

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

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

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

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

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

VARIABLE-LENGTH RECORDS

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

Variable-Length Records -- Format V

Format V provides for (1) variable-length records, (2) variable-length record segments, each of which describes its own characteristics, and (3) variable-length blocks of such records or record segments. The control program performs length checking of the block and uses the record or segment length information in blocking and deblocking. The first four bytes of each record, record segment, or block are a descriptor word containing control information. You must allow for these additional four 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 four-byte field which describes the block. The first two bytes specify the block length ('LL') -- four bytes for the BDW plus the total length of all records or segments within the block. This length can be from 8 to 32,760 bytes or, when using WRITE with tape, from 18 to 32,760. The third and fourth bytes are reserved for future system use and must be zero. 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 four-byte field describing the record. The first two bytes contain the length ('*ll*') of the logical record (including the four-byte RDW). The length can be from 5 to 32,756. For information about processing a sequential data set, see "Data Format -- Device Type Considerations." All bits of the third and fourth bytes must be zero as other values are used for spanned records. For output, you must provide the RDW. For input, the operating system provides the RDW except in data mode (spanned records). In data mode, the system passes the record length to the user in the logical record length field (DCBLRECL) of the data control block. The optional control character (C) may be specified as the fifth byte of each record and must be followed by at least one byte of data. The RDW and the control character, if specified, are not punched or printed.

Figure 3 shows blocked and unblocked variable-length records without the spanning feature.

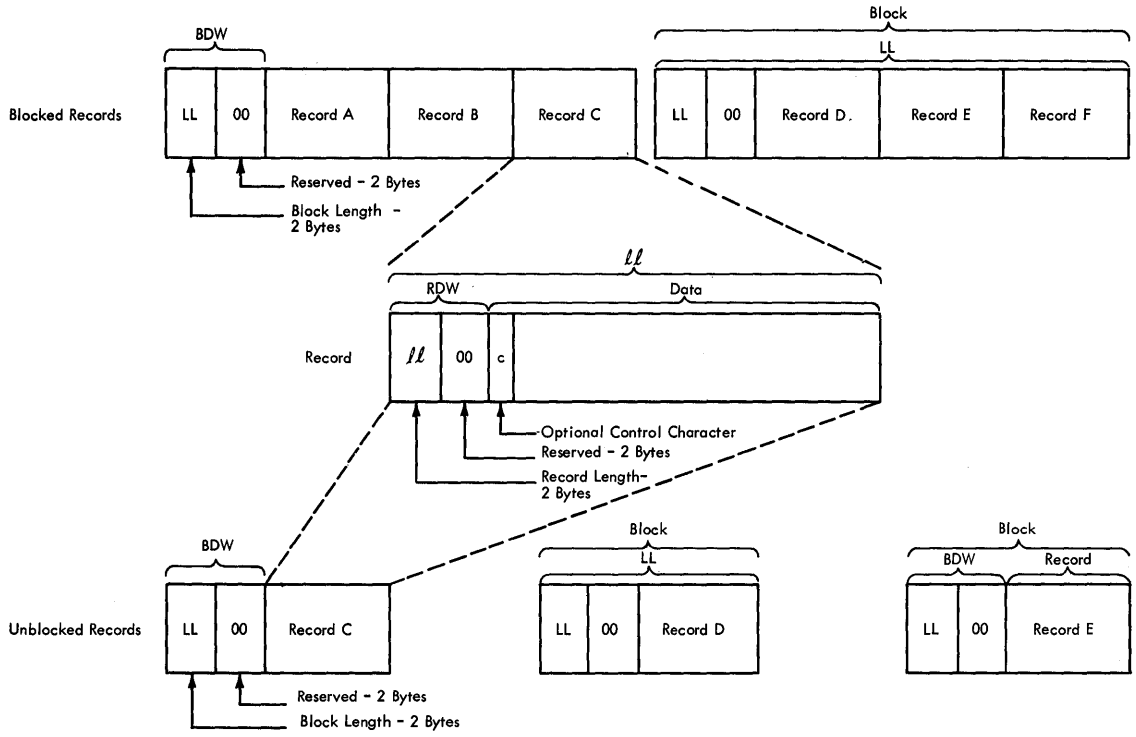
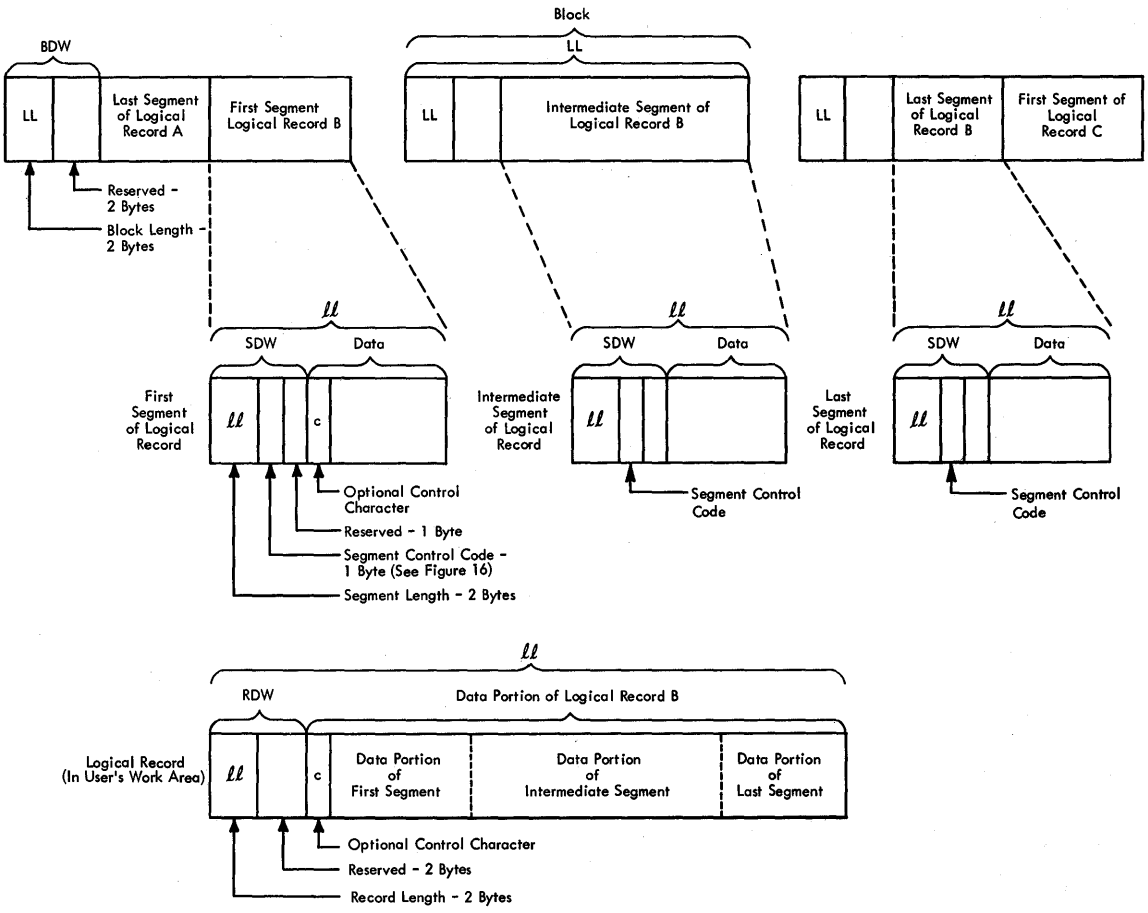


Figure 3. Variable-Length Records

SPANNED VARIABLE-LENGTH RECORDS (SEQUENTIAL ACCESS METHODS): The spanning feature of the queued and basic sequential access methods enables the user to create and process variable-length logical records which are larger than one physical block and/or to pack blocks with variable-length records. This is done by splitting the records into segments so that they can be written into more than one block, as shown in Figure 4.



Note: Not All Segment and Block Combinations are Represented

Figure 4. Spanned Variable-Length Records

When spanning is specified for blocked records, the system tries to fill all blocks. For unblocked records, a record which is 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 which 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 using QSAM.) A block can contain a combination of records and record segments but not multiple segments of the same record. When records are added to or deleted from a data set, or when the data set is processed again with different block- or record-size parameters, the record segmenting will change.

Segment Descriptor Word: Each record segment consists of a segment descriptor word (SDW) followed by the data. The segment descriptor word, similar to the record descriptor word, is a four-byte field which describes the segment. The first two bytes contain the length ('ll') of the segment including the four-byte SDW. The length can be from 5 to 32,756 bytes or, when using WRITE with tape, from 18 to 32,756 bytes. The third byte of the SDW contains the segment control code, which specifies the relative position of the segment in the logical record. The segment control code is in the rightmost two bits of the byte. The

segment control codes are shown in Figure 5. The remaining bits of the third byte and all of the fourth byte are reserved for future system use and must be zero.

Binary Code	Relative Position of Segment
00	Complete logical record.
01	First segment of a multi-segment record.
10	Last segment of a multi-segment record.
11	Neither first nor last segment of a multi-segment record.

Figure 5. Segment Control Codes

The SDW for the first segment replaces the RDW for the record after the record has been segmented. The SDW may be built by the user or the system depending on which mode of processing is used. In the basic sequential access method, the user must create and interpret the spanned records himself. In the queued sequential access method move mode, complete logical records, including the RDW, are processed in the user's work area. GET consolidates segments into logical records and creates the RDW. PUT forms segments as required and creates the SDW for each segment. Data mode is similar to move mode but allows reference only to the data portion of the logical record in the user's work area. The logical record length is passed to the user through the DCBLRECL field of the data control block. In the locate mode, both GET and PUT process one segment at a time. However, in the locate mode, if the user provides his own record area using the BUILDRCD macro instruction or if he asks the system to provide his record area by specifying BFTEK=A, GET, PUT, and PUTX process one logical record at a time. A logical record spanning three or more volumes cannot be processed when the data set is opened for update.

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

SPANNED VARIABLE-LENGTH RECORDS (BASIC DIRECT ACCESS METHOD): The spanning feature of the basic direct access method (BDAM) enables the user to create and process variable-length unblocked logical records that are longer than one track. The feature also enables the user to pack tracks with variable-length records by splitting the records into segments. These segments can then be written onto more than one track, as shown in Figure 6.

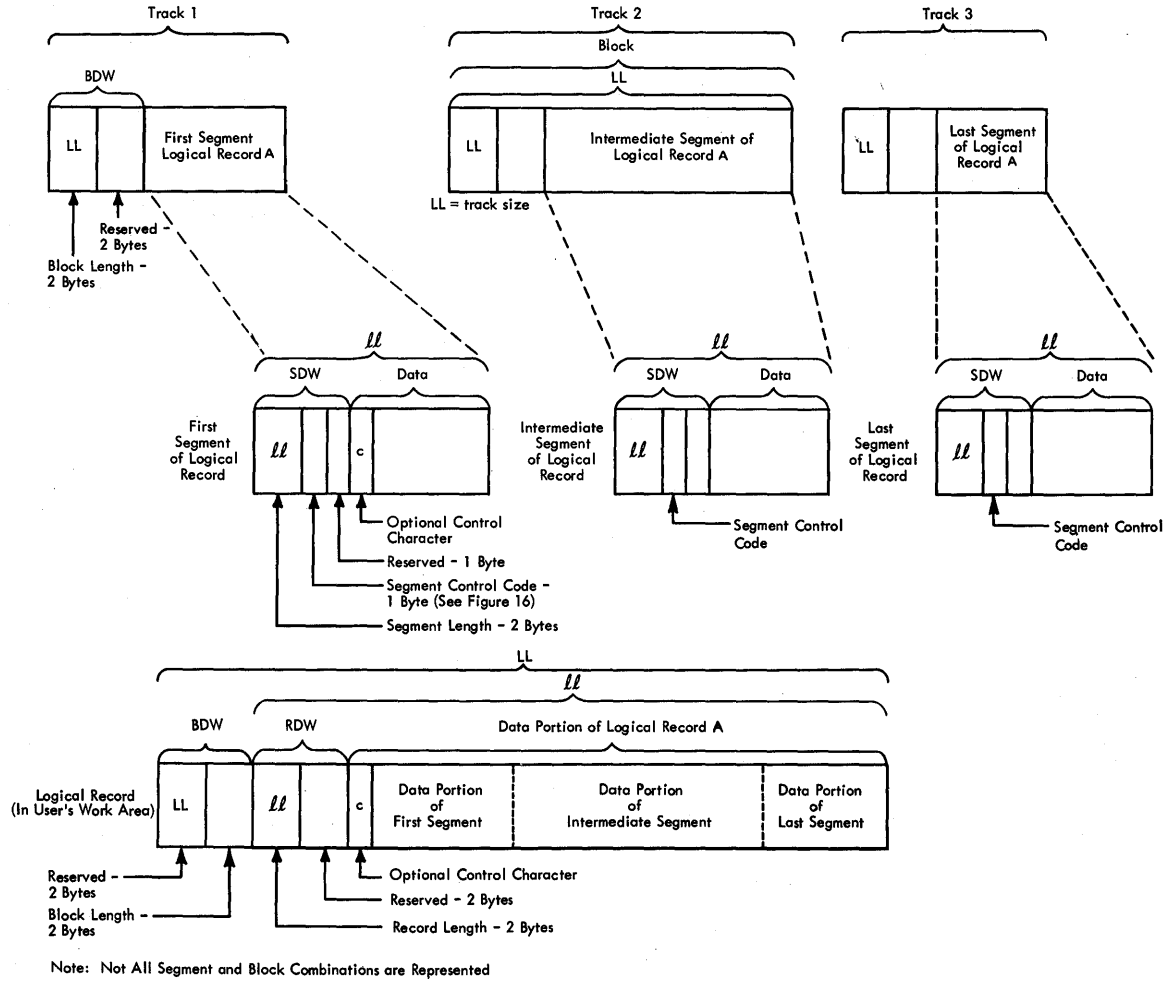


Figure 6. Spanned Variable-Length Records for BDAM Data Sets

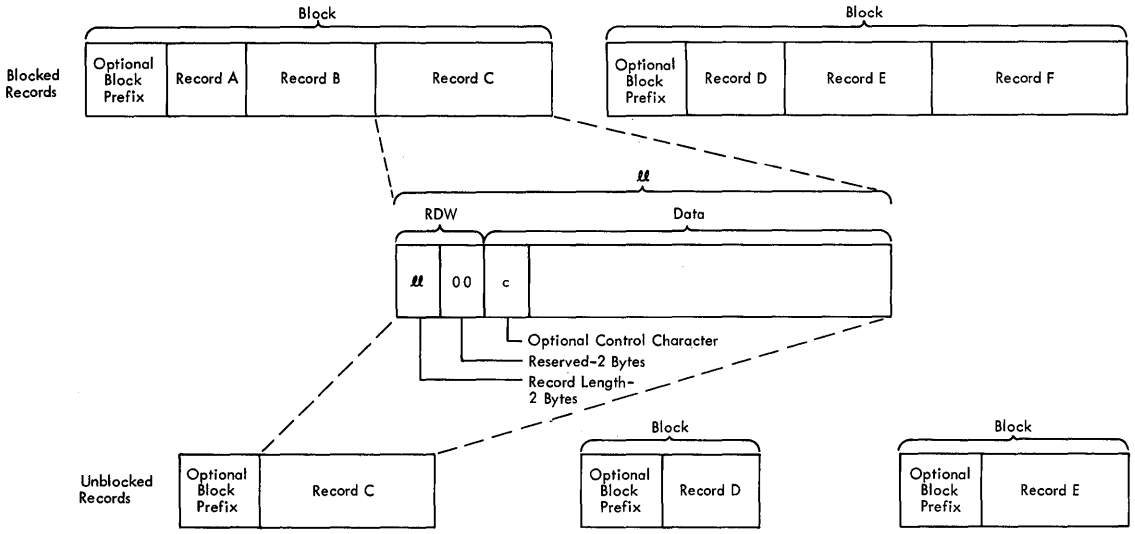
When you specify spanned, unblocked record format for the basic direct access method and when a complete logical record cannot fit on the track, the system tries to fill the track with a record segment. This aspect of spanning means that the maximum record length of a data set is not restricted by block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

Variable-Length Records -- Format D

For ASCII tapes, variable-length records must be format D. Format D records are the same as format V records, with the following exceptions:

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

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



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

Figure 7. Variable-Length Records for ASCII Tapes

The block prefix can vary in length from 0 to 99 bytes but must remain 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. If the block prefix is present, it precedes the RDW.

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

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

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

The block prefix can contain any data you want, but you must avoid using data types such as binary, packed decimal, floating-point, and any others that cannot be translated into ASCII. For format D records, the only time the block prefix can contain binary data is when you have coded BUFOFF=(L), which tells data management that the prefix is a BDW. Unlike the block prefix, the RDW must always be in binary.

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

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

UNDEFINED-LENGTH RECORDS

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

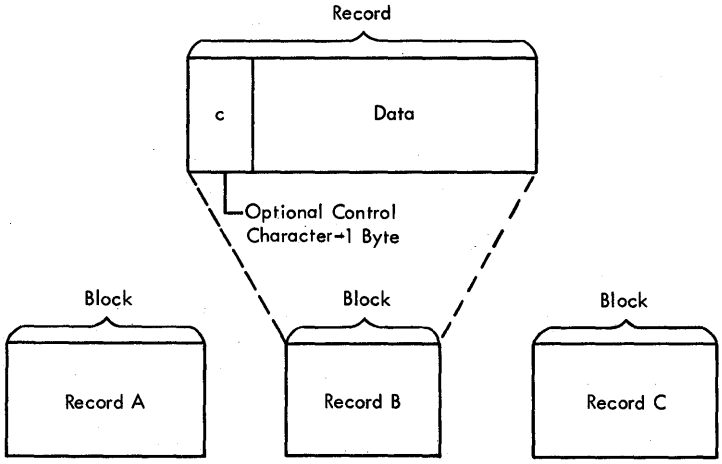


Figure 8. Undefined-Length Records

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

Figure 9 shows the format of undefined-length records for ASCII tapes and where control characters and block prefixes go if present.

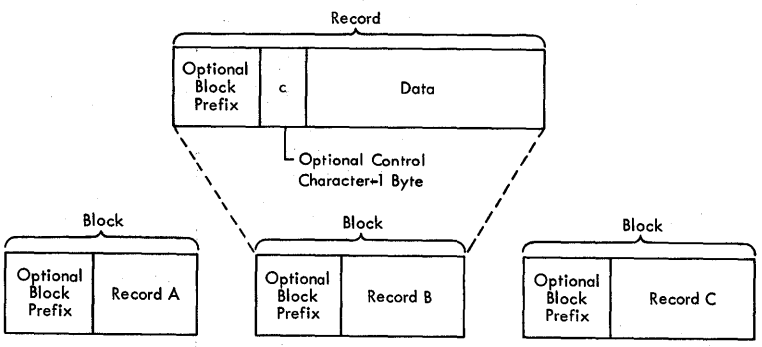


Figure 9. Undefined-Length Records for ASCII Tapes

CONTROL CHARACTER

You may specify in the DD statement, the DCB macro instruction, or the data set label that an optional control character is part of each record in the data set. The one-byte character is used to indicate a carriage control channel when the data set is printed or a stacker bin

when the data set is punched. Although the character is a part of the record in storage, it is never printed or punched. For that reason, buffer areas must be large enough to accommodate the character. If the immediate destination of the record is a device, such as disk, that does not recognize the control character, the system assumes that the control character is the first byte of the data portion of the record. If the destination of the record is a printer or punch and you have not indicated the presence of a control character, the system regards it as the first byte of data. A list of the control characters is in Appendix B.

Direct Access Device Characteristics

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

Although direct access devices differ in physical appearance, capacity, and speed, they are functionally and logically similar in terms of data recording, checking, format, and programming. The recording surface of each volume is divided into many tracks, each defined as the circumference of the recording surface. The tracks are arranged concentrically; their number and capacity varies with the device. Each device has some type of access mechanism, containing a number of read/write heads that transfer data as the recording surface rotates past.

The logical arrangement of related tracks is vertical rather than horizontal. As shown in Figure 10, a 1316 cylinder is comprised of ten tracks, which is equal to the number of recording surfaces. Because there are 203 tracks per disk, there are 203 vertical cylinders of ten tracks each.

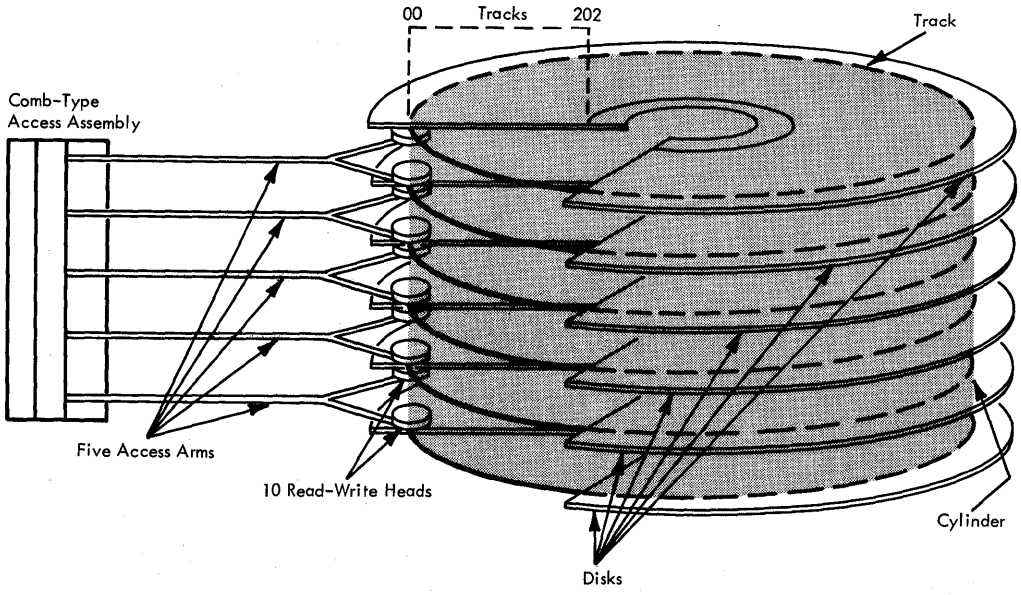


Figure 10. 1316 Disk Pack

TRACK FORMAT

Information is recorded on all direct access volumes in a standard format. In addition to device data, each track contains a track descriptor record ("capacity record" or R0), and data records. As shown in Figure 11, there are two possible data record formats -- Count-Data and Count-Key-Data -- only one of which can be used for a particular data set.

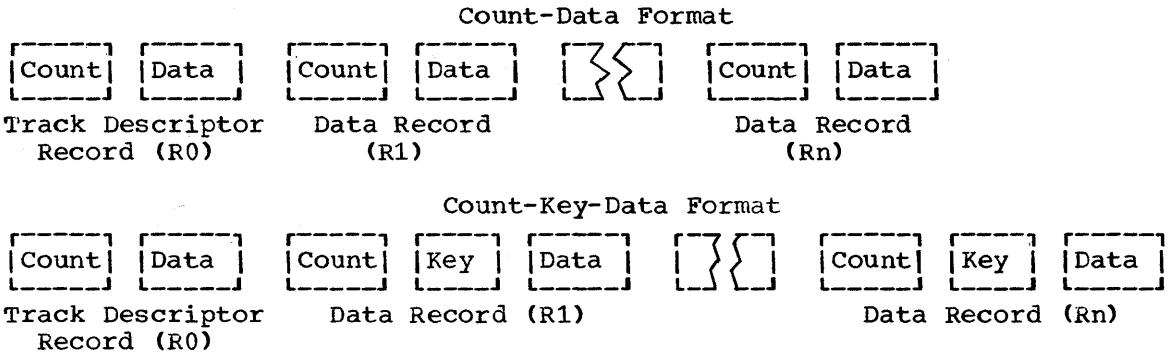


Figure 11. Direct Access Volume Track Formats

In addition to device data, the count area contains eight bytes that identify the location of the record in terms of the 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 identifies the following data area in terms of a part number, account number, sequence number, etc. In some cases, records are written with keys so that they can be located quickly.

TRACK ADDRESSING

There are two types of addresses that can be used to store and retrieve data on a direct access volume: actual and relative. The only real advantage of using actual addresses is the reduction in time required to convert from relative to actual address 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 one-byte binary number specifying the relative location of an entry in a data extent block (DEB). The data extent block is created by the system when the data set is opened. Each extent entry describes a set of contiguous tracks allocated for the data set.

BBCCHH
: is three two-byte binary numbers specifying the cell (bin), cylinder, and head number for the record, i.e., its track address. The cylinder and head numbers are recorded in the count area for each record.

R
is a one-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: There are two kinds of relative addresses that can be used to refer to records in a direct access data set: relative block address or relative track address.

The relative block address is provided as a three-byte binary number that indicates the position of the block in relation to the first block of the data set. Allocation of noncontiguous tracks does not affect the number. Therefore, the first block of a data set always has a relative block address of zero.

The relative track address is provided in the form TTR, where:

TT
is a two-byte binary number specifying the position of the track in relation to the first track allocated for the data set. The TT for the first track is zero. Allocation of noncontiguous tracks does not affect the number.

R
is a one-byte binary number specifying the number of the block in relation to the first block on the track TT. The first block of data on a track has a record value of one.

TRACK OVERFLOW

If the record overflow feature is available for the direct access device being used, you can reduce the amount of unused space on the volume by specifying the track overflow option in the DD statement or the DCB macro instruction associated with the data set. If the option is used, a block that does not fit on the track is partially written on that track and continued on the next available track. Each segment of an overflow block (the portion written on one track) 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 option is not used, blocks are not split between tracks.

Although a block can begin on one track and continue on the next, it cannot be continued on a noncontiguous track or from one separately allocated area to another.

WRITE VALIDITY CHECK

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

Interface With the Operating System

You must describe the characteristics of a data set, the volume on which it resides, and its processing requirements before processing can begin. During execution, the descriptive information is made available to the operating system in the data control block. A data control block 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, or a data set label. In addition, you can provide or modify some of the information during execution by storing the pertinent data in the appropriate field of the data control block. The specifications needed for input/output operations are supplied during the initialization procedures of the OPEN macro instruction. Therefore, the pertinent data can be provided when your job is to be executed rather than when you write your program (see Figure 12).

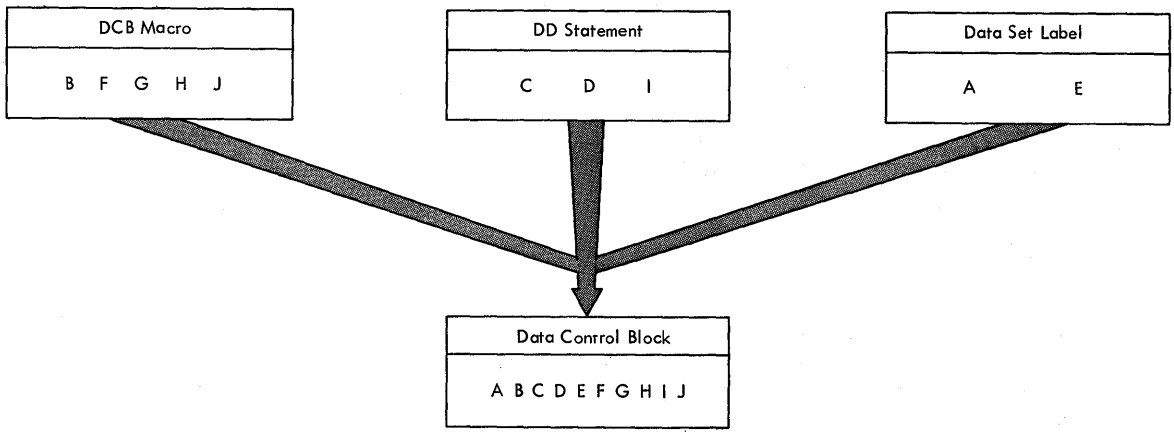


Figure 12. Completing the Data Control Block

When the OPEN macro instruction is executed, the open routine performs four primary functions:

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

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

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

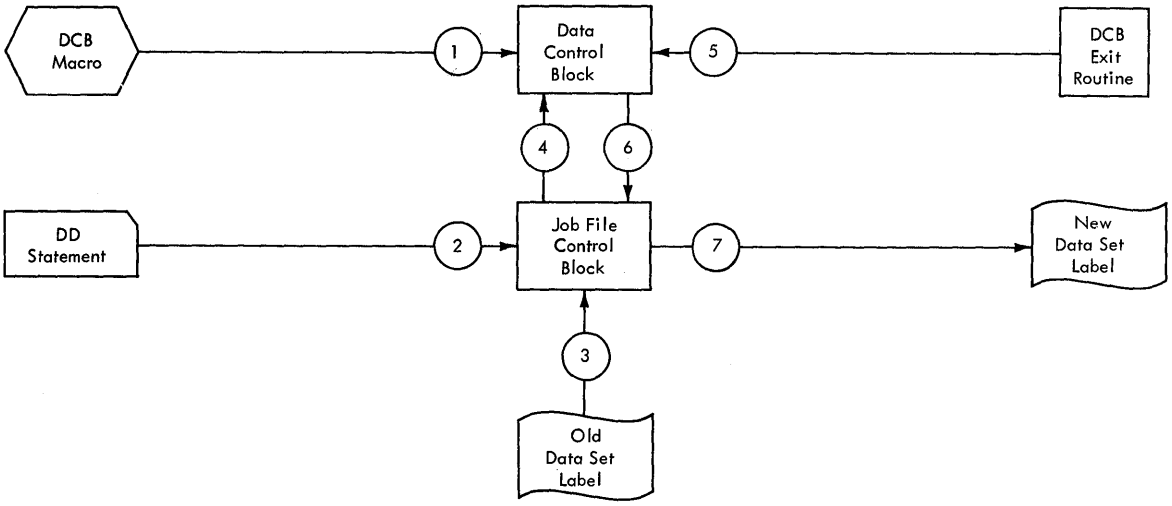


Figure 13. Source and Sequence for Completing the Data Control Block

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

DATA SET DESCRIPTION

For each data set you are going to process there must be a corresponding data control block and data definition statement. The characteristics of the data set and device-dependent information can be supplied by either source. In addition, the DD statement must supply data set identification, device characteristics, space allocation requests, and related information. The logical connection between a data control block and a DD statement is made by specifying the name of the DD statement in the DCB macro instruction (DDNAME), or by completing the field yourself before opening the data set.

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

Data Set Organization (DSORG): specifies the organization of the data set as physical sequential (PS), indexed sequential (IS), partitioned (PO), or direct (DA). If the data set contains location-dependent information (that is, absolute rather than relative addresses), it must be marked as unmovable, such as PSU. You must specify the data set organization in the DCB macro instruction. When creating or processing an indexed sequential organization data set or creating a direct organization data set, you must also specify DSORG in the DD statement.

Record Format (RECFM): specifies the characteristics of the records in the data set as fixed-length (F), variable-length (V), or undefined-length (U). Blocked records are specified as being FB or VB. Track overflow can be requested, for example, FBT.

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

Block Size (BLKSIZE): specifies the maximum length, in bytes, of a block. If the records are format F, the block size must be an integral multiple of the record length, including the key length, except for SYSOUT data sets. (See "Writing a SYSOUT Data Set" in Section II, Part 3, of this book.) If the records are format V, the block size must be the maximum block size. If records are unblocked, the block size must be four bytes greater than the record length (LRECL). When spanned variable-length records are specified, the block size is independent of the record length (LRECL).

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

Data Definition Name (DDNAME): is the name of the DD statement and provides a logical relationship to the data control block that specifies the same ddname.

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

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

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

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

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

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

Data Set Label (LABEL): indicates the type and contents of the label or labels associated with the data set. The operating system verifies standard labels (SL) or standard user labels (SUL). Nonstandard labels

(NSL) can be specified only if your installation has incorporated into the operating system routines to write and process nonstandard labels.

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

PROCESSING PROGRAM DESCRIPTION

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

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

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

Exits to Special Processing Routines: The DCB macro instruction can be used to identify the location of:

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

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

Exit Routine	When Available	Where Specified
End-of-Data-Set	No more sequential records or blocks are available	EODAD operand
Error Analysis	After an uncorrectable input/output error	SYNAD operand
Standard User Label (physically sequential or direct organization.)	Opening and closing or reaching the end of a data set, and when changing volumes.	EXLST operand and exit list
Data Control Block	Opening a data set	EXLST operand and exit list
End-of-Volume	When changing volumes	EXLST operand and exit list
Block Count	After unequal block count compare by EOV	EXLST operand and exit list

Figure 14. Data Management Exit Routines

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

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

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

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

- The contents of the general registers.
- The data event control block (discussed in Part 2).

- The exceptional condition code.

- The standard status and sense indicators.

There is a special macro instruction, SYNADAF, that you can use to perform this function automatically. This macro instruction produces a descriptive error message that can be printed by a subsequent PUT or WRITE macro instruction.

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

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

- ACC -- accept the erroneous block.

- SKP -- skip the erroneous block.

- ABE -- abnormally terminate the task.

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

When you use READ/WRITE macro instructions, errors are detected when you issue a CHECK macro instruction. If you are processing a direct or sequential data set and you return to the control program from your SYNAD routine, the operating system regards that as an acceptance of the bad record. If you are creating a direct data set and you return to the control program from your SYNAD routine, your task will be abnormally terminated.

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

Your SYNAD routine can end by branching to another routine in your program, such as a routine that closes the data set. It can also end by returning control to the control program, which then returns control to the next sequential instruction (after GET, PUT, etc.) in your program. If your routine returns control, the conventions for saving and restoring registers are as follows:

- The SYNAD routine must preserve the contents of registers 13 and 14. If required by the logic of your program, the routine must also preserve the contents of registers 2 through 12. Upon return to your program, the contents of registers 2 through 12 will be the same as upon return to the control program from the SYNAD routine.

- The SYNAD routine must not use the save area whose address is in register 13, because this area is used by the control program. If the routine saves and restores registers, it must provide its own save area.

- If the SYNAD routine calls another routine or issues supervisor or data management macro instructions, it must provide a save area in the usual way or by means of a SYNADAF macro instruction. The SYNADAF macro instruction provides a save area for its own use, and then makes this area available to the SYNAD routine. Such a save area must be removed from the save area chain by issuing a SYNADRLS macro instruction before returning control to the control program.

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

More specifically, suppose that you are using QSAM to read and translate a paper tape data set and that you have specified in your DCB SYNAD=(address) and EROPT=ACC. Suppose also that the third record of the data set has a parity error. When you issue a GET request for the second record, the system translates that record into your buffer and, as a result of the error in the third record, passes control to your SYNAD routine. Because you specified the accept option, the system returns control to your program after your SYNAD error analysis routine completes its processing. When you issue a GET request for the third record, that record is translated into your buffer as follows:

- The system translates the characters, up to the character in error, into your buffer.

- The system moves the character in error into your buffer without translating it.

- The system translates the remaining characters of the record into your buffer.

Exit List (EXLST): specifies the address of special processing routines. An exit list must be created if user label, data control block, end-of-volume, or block count exits are used.

The exit list is constructed of four-byte entries that must be aligned on fullword boundaries. The exit routine type is specified by a code in the high-order byte, and the address of the routine is specified in the three low-order bytes. Codes and addresses for the exit routines are shown in Figure 15.

You can activate or deactivate any entry in the list by placing the required code in the high-order byte. Care must be taken, however, so as not to destroy the last entry indication. The list will be scanned from top to bottom by the operating system. The first active entry found with the proper code will be selected.

Routine Type	Hexadecimal Code	3-Byte Routine Address - Purpose
Inactive entry	00	Ignored; the entry is not active
Input header label	01	Process a user input header label
Output header label	02	Create a user output header label
Input trailer label	03	Process a user input trailer label
Output trailer label	04	Create a user output trailer label
Data control block exit	05	Data control block exit routine
End-of-volume	06	End-of-volume exit routine
User totaling	0A	Pointer to user's totaling area
Block count exit	0B	Block count unequal exit routine
Defer input trailer label	0C	Defer processing of a user input trailer label from EOD until CLOSE
Defer nonstandard input trailer label	0D	Defer processing a nonstandard input trailer label on magnetic tape unit from EOD until CLOSE (no exit routine address)
Last entry	80	Last entry in list. This code can be specified with any of the above but must always be specified with the last entry.

Figure 15. Format and Contents of an Exit List

The list can be shortened during execution by setting the high-order four bits to the hexadecimal value 8. The list can be extended by setting the high-order four bits to zero.

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

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

The conventions for saving and restoring registers are as follows:

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

management macro instructions, it must provide the address of a new save area in register 13.

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

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

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

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

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

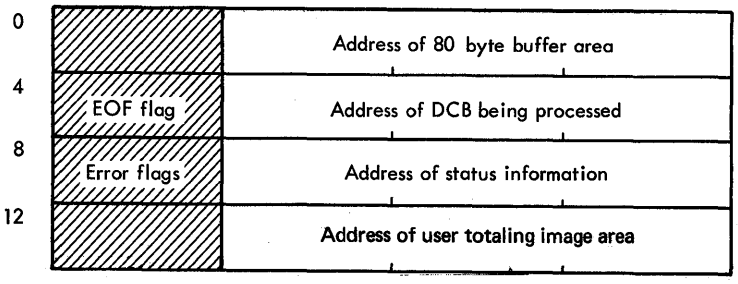


Figure 16. Parameter List Passed to User Label Routine

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

control program, which writes the label. When an input trailer label routine receives control, the EOF flag (high-order byte of the second entry in the parameter list) will be as follows:

```
bit 0 = 0:  Entered at end-of-volume.
bit 0 = 1:  Entered at end-of-file.
bits 1-7 :  Reserved.
```

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

```
bit 0 = 1   : Uncorrectable I/O error.
bit 1 = 1   : Error occurred when writing updated label.
bits 2-7    : Reserved.
```

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

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

Routine Type	Return Code	System Response
Input header or trailer label	0	Normal processing is resumed. If there are any remaining labels in the label group, they are ignored.
	4	The next user label is read into the label buffer area and control is returned to the exit routine. If there are no more labels in the label group, normal processing is resumed.
	8	The label is written from the label buffer area and normal processing is resumed.*
	12	The label is written from the label area, then the next label is read into the label buffer area and control is returned to the label processing routine. If there are no more labels, processing is resumed.*
Output header or trailer label	0	Normal processing is resumed; no label is written from the label buffer area.
	4	User label is written from the label buffer area. Normal processing is resumed.
	8	User label is written from the label buffer area. If less than eight labels have been created, control is returned to the exit routine, which then creates the next label. If eight labels have been created, normal processing is resumed.
*Only for a physically sequential data set opened for UPDATE or a direct data set opened for UPDATE or OUTPUT.		

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

You can create user labels only for data sets on magnetic tape volumes with IBM standard labels or American National Standards Institute (ANSI) labels or for data sets on direct access. When you specify IBM standard or ANSI labels and user labels in a DD statement by specifying LABEL=(,SUL) or LABEL=(,AUL) and there is an active entry in the exit list, a label exit is always taken. A label exit may be taken when an input data set does not contain user labels, or when no user label track has been allocated for writing labels on a direct access volume. In either case, the appropriate exit routine is entered with the buffer area address parameter set to zero. On return from the exit routine, normal processing is resumed; no return code is necessary.

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

- When the data set is opened, header label exits are taken, except when the data set already exists and DISP=MOD is coded in the DD statement. In the latter case, the volume is positioned to the end of the data set, and input trailer label exits are taken.

- When end-of-volume is reached, trailer label exits are taken; header label exits are taken after volume switching. Input trailer label exits are not taken, however, if you force end-of-volume by issuing an FEOV macro instruction.

- When end-of-data is reached, input trailer label exits are taken before the EODAD exit, unless the data control block (DCB) exit list indicates defer input trailer label processing. When an output data set is closed, output trailer label exits are taken.

- When end-of-data is reached for a direct access data set and the data control block (DCB) exit list indicates defer input trailer label processing, the system changes the 0C to 0D. When the Close routine has finished processing, the system changes the code back to 0C.

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

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

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

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

User Totaling: (BSAM, QSAM only) When creating or processing a data set with user labels, you may develop control totals for each volume of the data set and store this information in your user labels. For example, a control total that was accumulated as the data set was created can be stored in your user label and later compared with a total accumulated while processing the volume. The user totaling facility assists you by synchronizing the control data which you create with records physically written on a volume. For an output data set without user labels, you can also develop a control total which will be available to your end-of-volume routine.

To request this facility, you must specify OPTCD=T in the DCB macro instruction or in the DCB parameter of the DD statement. The area in which you accumulate the control data, the user's totaling area, must be identified to the control program by an X'0A' entry in the data control block (DCB) exit list.

The user's totaling area, an area in storage that you provide, must begin on a halfword boundary and be large enough to contain your accumulated data plus a two-byte length field. The length field must be the first two bytes of the area and specify the length of the entire area. A data set for which you have specified user totaling (OPTCD=T)

will not be opened if either the totaling area length or the address in the exit list is zero, or if there is no X'0A' entry in the exit list.

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

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

When variable-length records are specified with the totaling facility for user labels, special considerations are necessary. Since the control program determines whether a variable-length record will fit in a buffer after a PUT or a WRITE has been issued, the total you have accumulated may include one more record than is actually written on the volume. In the case of variable-length spanned records, the accumulated total will include the control data from the volume-spanning record although only a segment of the record is on that volume. However, when processing such a data set, the volume-spanning record or the first record on the next volume will not be available to you until after the volume switch and user label processing is completed. Thus the totaling information in the user label may not agree with that developed while processing the volume. One way you can resolve this situation is to maintain, when you are creating a data set, control data pertaining to each of the last two records and include both totals in your user labels. Then the total related to the last complete record on the volume and the volume-spanning record or the first record on the next volume would be available to your user label routines. During subsequent processing of the data set, your user label routines can determine if there is agreement between the generated information and one of the two totals previously saved.

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

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

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

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

The end-of-volume exit routine can be used to take a checkpoint by issuing the CHKPT macro instruction, which is discussed in "Section 1: Supervisor Services". If the job step terminates abnormally, it can be restarted from this checkpoint. When the job step is restarted, the volume is mounted and positioned as upon entry to the routine. Note that restart becomes impossible if changes are subsequently made to the system SVC library (SYS1.SVCLIB). When the step is restarted, pointers to end-of-volume modules must be the same as when the checkpoint was taken.

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

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

The routine is entered during end-of-volume processing. The trailer label block count will be passed in register 0. The user may access the count field in the data control block by addressing the address passed in register 1 plus the proper displacement as given in the System Control Blocks manual. If the block count in the data control block differs from that in the trailer label when no exit routine is provided, the task is abnormally terminated.

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

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

Figure 18. System Response to Block Count Exit Return Code

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

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

For an end-of-volume condition, the EOV routine will pass control to the user's nonstandard input trailer label routine, whether or not this exit code is specified. For an end-of-data condition when this exit code is specified, the EOV routine does not pass control to the user's nonstandard input trailer label routine. Instead, the Close routine passes control to the user's routine.

MODIFYING THE DATA CONTROL BLOCK

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

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

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

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

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

Changing an Address in the Data Control Block: Example 1 illustrates how you can modify a field in the data control block. The DCBD macro instruction defines the symbolic name of each field.

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

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

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

Example 1. Modifying a Field in the Data Control Block

SHARING A DATA SET

A data set can be shared by all the tasks of a job step. If requested in the DD statement, a data set can be shared by all the tasks in the system. (Remember that there is only one task in a system with PCP.)

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

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

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

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

Part 2: Data Management Processing Procedures

Data Processing Techniques

The operating system allows you to concentrate your efforts on processing the records read or written by the data management routines. 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 can be defined as the combination of data set organization and the technique used to process the data. Data access techniques can be divided into two categories -- queued and basic.

QUEUED ACCESS TECHNIQUE

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

Because the operating system controls buffer processing, you can use as many I/O buffers as needed without reissuing GET/PUT macro instructions to fill or empty buffers. Usually, more than one input block is in main storage at any given time to prevent I/O operations from delaying record processing.

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

GET -- Retrieve a Record

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

In an operating system with the time sharing option (TSO), you can issue a GET macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, then the GET results in a TGET macro instruction, which retrieves data from the terminal.

PUT -- Write a Record

The PUT macro instruction places a record into an output data set. Like the GET macro instruction, it operates in a logically sequential and device-independent manner. As required, the PUT macro instruction schedules the emptying of output buffers, blocks records, and handles output error correction procedures. For sequential data sets, it also initiates automatic volume switching and label creation, and also segments records for spanning. If you specify OPTCD=Q in the DCB, PUT causes output to be translated from EBCDIC to ASCII.

If the PUT macro instruction is directed to a card punch or printer, the system automatically adjusts the number of records or record segments per block of format F or 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 one card or one print line. Record size may be greater than block size in this case.

In an operating system with the time sharing option (TSO), you can issue a PUT macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, then the PUT results in a TPUT macro instruction, which sends data to the terminal.

PUTX -- Write an Updated Record

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

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

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

In an operating system with the time sharing option (TSO), you can issue a PUTX macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, then the PUTX results in a TPUT macro instruction, which sends data to the terminal.

BASIC ACCESS TECHNIQUE

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

buffering or synchronized scheduling, macro instructions are provided to help you program these functions.

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

READ -- Read a Block

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

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

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

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

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

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

Direct
- D - use the direct access method.
- I - locate the block using a block identification.
- K - locate the block using a key.
- F - provide device position feedback.
- X - maintain exclusive control of the block.
- R - provide next address feedback.
- U - next address can be a capacity record or logical record, whichever occurred first.

In an operating system with the time sharing option (TSO), you can issue a READ macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, then the READ results in a TGET macro instruction, which retrieves data from the terminal.

WRITE -- Write a Block

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

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

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

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

Direct
- SD - write a dummy fixed-length record.
- SZ - write a capacity record (R0). The system supplies the data, writes the capacity record, and advances to the next track.
- D - use the direct access method.
- I - search argument identifies a block.
- K - search argument is a key.
- A - add a new block.
- F - provide record location data (feedback).
- X - release exclusive control.

In an operating system with the time sharing option (TSO), you can issue a WRITE macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, then the WRITE results in a TPUT macro instruction, which sends data to the terminal.

CHECK -- Test Completion of Read/Write Operation

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

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

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

In an operating system with the time sharing option (TSO), you can issue a CHECK macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, then the CHECK results in the following:

- When used after a READ macro instruction, the CHECK macro instruction takes the end-of-data exit (specified by the EODAD parameter of the DCB) when the input data set reaches end of file. If the end-of-file condition does not occur, the CHECK macro instruction is treated as a NOP instruction.

- When used after a WRITE macro instruction, the CHECK macro instruction is treated as a NOP instruction.

WAIT -- Wait for Completion of a Read/Write Operation

When processing a data set, you can test for completion of any read or write operation by issuing a WAIT macro instruction. The input/output operation will be synchronized with processing, but the DECB will not be checked for errors or exceptional conditions, nor will end-of-volume procedures be initiated. These functions must be tested for and performed by your program.

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

Data Event Control Block (DECB)

A data event control block is a 16- to 32-byte area reserved by each READ/WRITE macro instruction. It contains control information and pointers to standard status indicators. It is described in detail in the Supervisor and Data Management Macro Instructions publication.

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

ERROR HANDLING

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

SYNADAF -- Perform SYNAD Analysis Function

The SYNADAF macro instruction analyzes the status, sense, and exceptional condition code data that is available to your error analysis routine. It produces a descriptive 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 it can be written as a fixed-length record by omitting the block and record length fields that precede the message text.

The text of the message is 120 characters in length, and begins with a field of 36 or 42 blanks; you can use the blank field to add your own

remarks to the message. Following is a typical message with the blank field omitted:

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

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

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

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

SYNADRLS -- Release SYNADAF Message and Save Areas

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

ATLAS -- Perform Alternate Track Location Assignment

The ATLAS macro instruction enables your program to recover from permanent input/output errors when processing a data set in direct access storage. After a data check, or in certain missing address marker conditions, you can issue ATLAS to:

- Assign an alternate track to replace the error track.
- Transfer data from the error track to the alternate track.

Use of the ATLAS macro instruction requires a knowledge of channel programming. For this reason, a detailed description of the macro instruction and its use is included in the System Programmer's Guide.

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

TIME SHARING OPTION (TSO) TERMINAL ACCESS

TSO provides two macro instructions, TGET and TPUT, for you to use to transfer data between a terminal and your program running in a TSO region. The TGET macro instruction retrieves data from a terminal, and the TPUT macro instruction sends data to a terminal.

TGET -- Retrieve a Record From The Terminal

The TGET macro instruction transfers data that has been entered at a terminal to input buffers that are defined in the foreground program, a

program running in a time sharing region. If an input buffer is not large enough to contain an entire input record, a return code is set, and the next TGET instruction issued by the foreground program accesses the remaining record segment. (The return codes are discussed in the Supervisor and Data Management Macro Instructions manual.) Note that no DCB is required with TGET.

A TGET macro instruction issued by a program running in the background (not in a time sharing region) is treated as a NOP instruction.

TPUT -- Write a Record to The Terminal

The TPUT macro instruction is issued by a foreground program to send data to a terminal. If the terminal line size is not large enough to contain a record, the record is entered on the number of lines necessary to complete it.

The TPUT macro instruction can also be used to send data to terminals other than the terminal identified with your program. This is done by specifying, in the operand field of the TPUT statement, the TJID (terminal job identifier) of another terminal currently in use. Note that no DCB is required with TPUT.

A TPUT macro instruction issued by a program running in the background is treated as a NOP instruction.

SELECTING AN ACCESS METHOD

Access methods are identified primarily by the data set organization to which they apply. For instance, we speak of a basic access method for direct organization (BDAM). 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 is created using the basic access method for sequential organization (BSAM). It is processed using the basic direct access method (BDAM). If the queued access technique is used to process a sequential data set, the access method is referred to as QSAM.

The basic access methods are used for all data organizations, while the queued access methods apply only to sequential and indexed sequential data sets as shown in Figure 19.

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

Figure 19. Data Management Access Methods

It is possible to directly control an I/O device while processing any data organization without using a specific access method. The execute channel program (EXCP) macro instruction uses the system functions 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 instruction are in the System Programmer's Guide.

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 it is initiated for execution. Initiation is accomplished by issuing the OPEN macro instruction. After all data control blocks have been completed, the system ensures that all required access method routines are loaded and ready for use and that all channel command word lists and buffer areas are ready.

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

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

This information is used by the system to allocate main storage space and load the appropriate routines. These routines, the CCW lists, and buffer areas created automatically by the system remain in main storage until the close routine signals that they are no longer needed by that data control block.

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

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

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

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

Notes:

1. Two or more data control blocks should never be opened concurrently for output to the same data set on a direct access device. This may result in the end-of-file record written by the CLOSE for one data control block overlaying data associated with another data control block.

2. Two or more data control blocks should never be opened concurrently using the same DDNAME. This is true for both input and output and especially important when using more than one access method. Any action on one DCB that alters the TIOT or JFCB affects the other DCB(s) and thus can cause unpredictable results.

3. If you want to use the same DD statement for two or more DCBs, you cannot specify parameters for fields in the first DCB and then be assured of obtaining the default parameters for the same fields in any subsequent DCB using the same DD statement. Therefore, unless the parameters of all DCBs using one DD statement are the same, you should use separate DD statements.

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

OPEN -- Initiate Processing of a Data Set

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

Processing Method: A data set can be processed as either input or output (INPUT, OUTPUT) or a combination of the two (INOUT, OUTIN -- BSAM only). If the data set resides on a direct access volume, records can be updated (UPDAT). Magnetic tape volumes can also be read backwards (RDBACK -- BSAM and QSAM only). If the processing method operand is omitted from the OPEN macro instruction, INPUT is assumed. The operand is ignored by BISAM; it must be specified as OUTPUT when using QISAM to create an indexed sequential data set. You can override the OPEN options INOUT and OUTIN at execution time by using the LABEL parameter of the DD card. Use of this facility is discussed in the Job Control Language Reference manual.

Simultaneous Opening of Data Sets: In Example 2, the data sets associated with three data control blocks are to be opened simultaneously, with the indicated options.

```
          OPEN  (TEXTDCB,,CONVDCB,(OUTPUT),PRINTDCB,(OUTPUT))
+         CNOP  0,4
+         BAL   1,*+16 LOAD REG1 W/LIST ADDR.
+         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
```

Example 2. Opening Three Data Control Blocks Simultaneously

Since no processing method operand is specified for TEXTDCB, the system assumes INPUT. Both CONVDCB and PRINTDCB are opened for output. No volume positioning options are specified; thus, the position indicated by the DD statement DISP parameter is used.

At execution time, the SVC 19 instruction passes control to the open routine, which then initiates the three data control blocks 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 data control block. The volume positioning that is to result from closing the data set can also be specified. Volume positioning options are the same as those that can be specified for end-of-volume conditions, as specified in the OPEN macro instruction or the DD statement. An additional volume positioning option, REWIND, is available and can be specified by the CLOSE macro instruction for magnetic tape volumes. REWIND positions the tape at the load point regardless of the direction of processing.

The operating system provides a temporary closing option, CLOSE (TYPE=T), for data sets being processed by BSAM. CLOSE (TYPE = T) causes the RLSE parameter on the DD card to be ignored. When the macro instruction is executed for data sets on magnetic tape or direct access volumes, the system processes labels and repositions the volume as required. However, the data control block maintains its open status. Processing of the data set can be continued at a later stage in your program without reissuing the OPEN macro instruction. Performance is thus improved significantly. Magnetic tape volumes will be repositioned either preceding the first data block or following the last data block of the data set. The presence of tape labels has no effect on repositioning.

Simultaneous Closing of Data Sets: In Example 3 In example 3 the data sets associated with three data control blocks are to be closed simultaneously.

```
          CLOSE (TEXTDCB,,CONVDCB,,PRINTDCB)
+         CNOP  0,4
+         BAL   1,*+16 BRANCH AROUND LIST
+         DC    AL1(0) OPTION BYTE
+         DC    AL3(TEXTDCB) DCB ADDRESS
+         DC    AL1(0) OPTION BYTE
+         DC    AL3(CONVDCB) DCB ADDRESS
+         DC    AL1(128)OPTION BYTE
+         DC    AL3(PRINTDCB) DCB ADDRESS
+         SVC   20 ISSUE CLOSE SVC
```

Example 3. Closing Three Data Control Blocks Simultaneously

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

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

End-of-Volume Processing

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

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

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

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

Multiple volume data sets can be specified in your DD statement whereby 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 are required than specified, the storage is obtained from any available volume of the same device type. If no device is available, your job is terminated.

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

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

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

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

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

LEAVE
: will position a labeled tape following the tapemark that follows the data set trailer label group; an unlabeled volume following the tapemark that follows the last block of the data set.

REREAD
: will position a labeled tape preceding the data set header label group; an unlabeled tape preceding the first block of the data set.

If the tape was last read backwards:

LEAVE
: will position a labeled tape preceding the data set header label group; an unlabeled tape preceding the first block of the data set.

REREAD
: will position a labeled tape following the tape mark that follows the data set trailer label group; an unlabeled tape following the tape mark that follows the last block of the data set.

FEOV -- Force End of Volume

The FEOV macro instruction directs the operating system to initiate end-of-volume processing before the physical end of the current volume

is reached. If another volume has been specified for the data set, volume switching takes place automatically. The volume positioning options REWIND and LEAVE are available.

The FEOV macro instruction can only be used when processing data sequentially, that is, using BSAM or QSAM.

In an operating system with the time sharing option (TSO), you can issue an FEOV macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, the FEOV is treated as a NOP instruction.

Buffer Acquisition and Control

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

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

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

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

BUFFER POOL CONSTRUCTION

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

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

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

In many applications, singleword or doubleword alignment of a block within a buffer is important. You can specify in the data control block that buffers are to start on either a doubleword or a fullword boundary that is not also a doubleword boundary (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 record.

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

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

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

BUILD -- Construct a Buffer Pool

When you know, prior to program assembly, both the number and 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 -- a predefined storage area, or an area of coding no longer needed, for example.

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

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

BUILDRCD -- Build a Buffer Pool and a Record Area

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

- It provides the logical record interface necessary for a sequential data set accessed by QSAM in the locate mode and having a record format of VS or VBS. Logical record interface, unlike segment interface, enables the user to access an entire logical record, not just a segment.

- It links a record area to the buffer control block by extending the buffer control block to twelve bytes. Thus, a spanned record can be assembled or segmented in the record area.

GETPOOL -- Get a Buffer Pool

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

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

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

Automatic Buffer Pool Construction

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

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

FREEPOOL -- Free a Buffer Pool

Any buffer pool assigned to a DCB either automatically by the OPEN macro instruction (except when dynamic buffer control is used) or explicitly by the GETPOOL macro instruction must be released before your program is terminated. The FREEPOOL macro instruction should be issued to release the main storage area as soon as the buffers are no longer needed. As a general rule, when you are using the queued access technique, an output data set should be closed first to ensure that all the records have been written out. However, when using exchange buffering or when processing an indexed sequential data set using the queued access technique, the buffer pool must not be released until all the data sets have been closed.

Constructing a Buffer Pool: The following examples illustrate several possible methods of constructing a buffer pool. The examples do not consider the method of processing or controlling the buffers in the pool.

```
         ...                                    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
```

Example 4. Constructing a Buffer Pool From a Static Storage Area

In Example 4, a static storage area named INPOOL is allocated during program assembly. The BUILD macro instruction, issued during execution, arranges the buffer pool into ten buffers, each 52 bytes long. Five buffers are assigned to INDCB and five to OUTDCB, as specified in the DCB macro instruction for each. The two data sets share the buffer pool because both specify INPOOL as the buffer pool control block. Notice that an additional eight bytes have been allocated for the buffer pool to contain the buffer pool control block.

```
          ...
          GETPOOL   INDCB,10,52          Construct a 10-buffer pool
          GETPOOL   OUTDCB,5,112         Construct a  5-buffer pool
          OPEN      (INDCB,,OUTDCB,(OUTPUT))
          ...
ENDJOB    CLOSE     (INDCB,,OUTDCB)
          FREEPOOL  INDCB                Release buffer pools after all
          FREEPOOL  OUTDCB                I/O is complete
          ...
          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,---
```

Example 5. Constructing a Buffer Pool Using GETPOOL and FREEPOOL

In Example 5, two buffer pools are constructed explicitly by the GETPOOL macro instructions. Ten input buffers are provided, each 52 bytes long, to contain one fixed-length record; five output buffers are provided, each 112 bytes long, to contain two blocked records plus an 8-byte count field (required by the Indexed Sequential Access Method). Notice that both data sets are closed before the buffer pools are released by the FREEPOOL macro instructions. The same procedure should be used if the buffer pools were constructed automatically by the OPEN macro instruction.

BUFFER CONTROL

There are four techniques that can be used 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. Both simple and exchange buffering are provided for the queued access technique. The basic access technique provides for either direct or dynamic buffer control.

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

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

- Move mode: The record is moved from an input buffer to your work area, or from your work area to an output buffer.
- Data mode (QSAM V format spanned records only): The same as the move mode except only the data portion of the record is moved.

- Locate mode: The record is not moved. Instead, the address of the next input or output buffer is placed in register 1.

  For QSAM format V spanned records, the record is not moved. Instead, if logical record interface has been requested by specifying BFTEK=A or by issuing the BUILDRCD macro instruction, the address returned in register 1 points to a record area where the spanned record is assembled or segmented.

- Substitute mode: The record is not moved. Instead, the address of the next input or output buffer is interchanged with the address of your work area.

Two processing modes of the PUTX macro instruction can be used in conjunction with a GET-locate macro instruction. The update mode returns an updated record to the data set from which it was read; the output mode transfers an updated record to an output data set. There is no actual movement of data in main storage. The processing mode must be specified in the MACRF parameter of the DCB macro instruction.

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

- Directly using the GETBUF macro instruction to retrieve a buffer constructed as described above. A buffer can then be returned to the pool using the FREEBUF macro instruction.

- Dynamically by requesting a dynamic buffer in your READ/WRITE macro instruction. This technique can be used only when processing an indexed sequential or direct organization data set. If you request dynamic buffering, the system will automatically provide a buffer each time a READ macro instruction is issued. The buffer is supplied from a buffer pool which is created by the system when the data control block of the data set is opened. The buffer will be released (returned to the pool) upon completion of a WRITE macro instruction when you are updating. If you do not update the record in the buffer and thus release the buffer when the record is written, the FREEDBUF macro instruction may be used. If you are processing an indexed sequential data set, the buffer is automatically released upon the next issuance of the READ macro instruction if there has been no intervening WRITE or FREEDBUF macro instruction issued.

Simple Buffering

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

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

- Processed in an input buffer and then moved to an output buffer (GET-locate, PUT-move/PUTX-output).

- Moved from an input buffer to an output buffer where it can be processed (GET-move, PUT-locate).

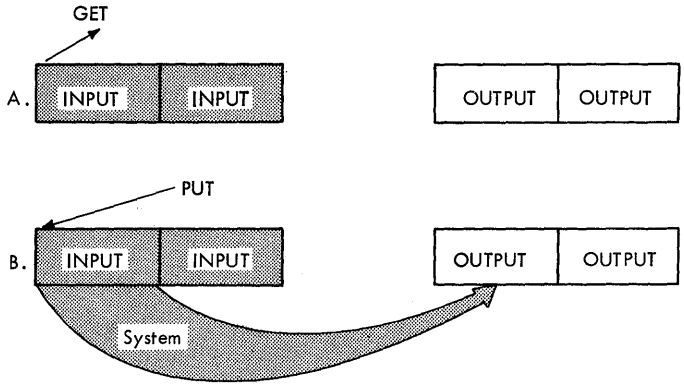
- Moved from an input buffer to a work area where it can be processed and then moved to an output buffer (GET-move, PUT-move).
- Processed in an input buffer and returned to the data set (GET-locate, PUTX-update).

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

Simple Buffering -- GET-locate, PUT-move/PUTX-output: The GET macro instruction (step A, Figure 20) locates the next input record to be processed. Its address is returned in register 1 by the system. The address is passed to the PUT macro instruction in register 0.

The PUT macro instruction (step B, Figure 20) specifies the address of the record in register 0. The system then moves the record to the next output buffer.

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




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

Figure 20. Simple Buffering (GL,PM)

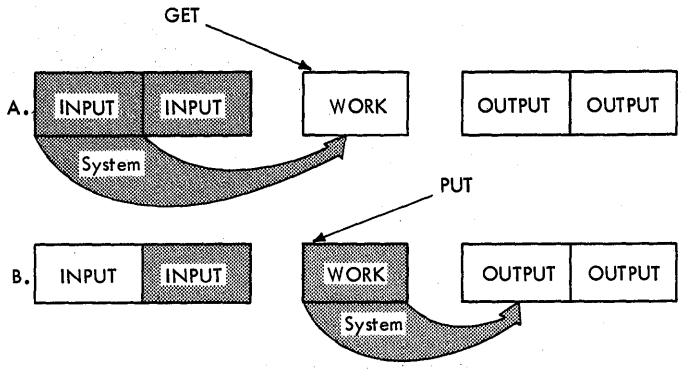
Simple Buffering -- GET-move, PUT-locate: The PUT macro instruction locates the address of the next available output buffer. Its address is returned in register 1 and is passed to the GET macro instruction in register 0.

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

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

Simple Buffering -- GET-move, PUT-move: The GET macro instruction (step A, Figure 21) specifies the address of a work area into which the system moves the next record from the input buffer.

The PUT macro instruction (step B, Figure 21) specifies the address of a work area from which the system moves the record into the next output buffer.




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

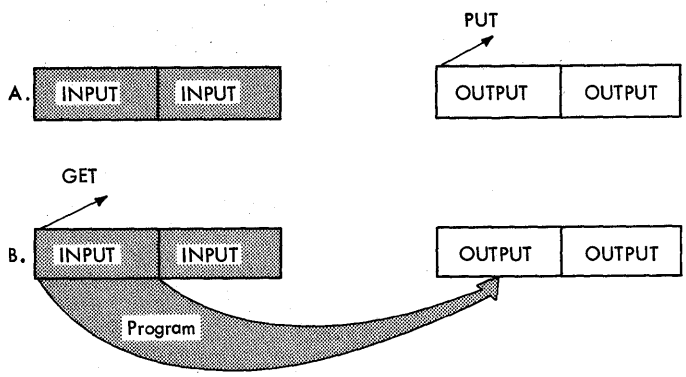
Figure 21. Simple Buffering (GM, PM)

Simple Buffering -- GET-locate, PUT-locate: The PUT macro instruction (step A, Figure 22) locates the address of the next available output buffer. The address is returned in register 1.

The GET macro instruction (step B, Figure 22) locates the address of the next input buffer. Its address is returned in register 1. You must then move the record from the input buffer to the output buffer. Processing can be done either before or after the move operation.

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

Note: If records other than format F are being moved, the length attribute of the MVC instruction must be changed as shown. If the record is more than 256 bytes, you will have to code a move routine to process the complete record.




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

Figure 22. Simple Buffering (GL, PL)

Exchange Buffering

The term "exchange" buffering refers to the relationship of segments within a buffer. All the segments in an exchange buffer are not necessarily contiguous in main storage, nor are they always associated

with the same data set. When the buffer pool is constructed, the system creates a channel command word (CCW) for each buffer segment in the buffer. This facility makes it possible to "exchange" the CCWs of different storage locations.

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

Why use exchange buffering? Because there is no need to move the record. This means a considerable savings in processing time. On the other hand, exchange buffering is of no advantage unless substitute mode or PUTX-output mode is used.

The implementation of exchange buffering during execution of your program depends on a number of factors:

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

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

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

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

To reopen a DCB that has been opened for exchange buffering, you must first do the following:

- Close all DCBs using the buffer pool associated with the DCB to be reopened.
- Issue a FREEPOOL macro instruction specifying the DCB to be reopened.

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

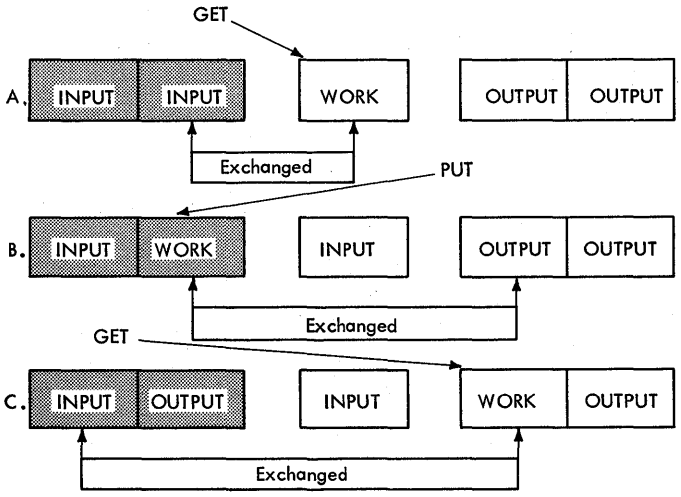
- Word alignment that does not correspond to the characteristics of the machine. If, for example, you expect to process your data on a System/360 model 65, 75, 85, or 91 or on a System/370, your record length should be a multiple of 16; on a model 50, a multiple of 8; on a model 40, a multiple of 4.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

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

Exchange Buffering -- GET-substitute, PUT-substitute: The GET macro instruction (step A, Figure 23) specifies the address of a work area. The work area address is exchanged for the address of the next input record returned in register 1. After processing, the address of the record is passed to the PUT macro instruction.

The PUT macro instruction (step B, Figure 23) specifies the address of the output record. The output record address is exchanged for the address of the next output buffer available for use as a work area. The work area address, returned in register 1, is passed to the GET macro instruction (step C, Figure 23) in register 0.

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

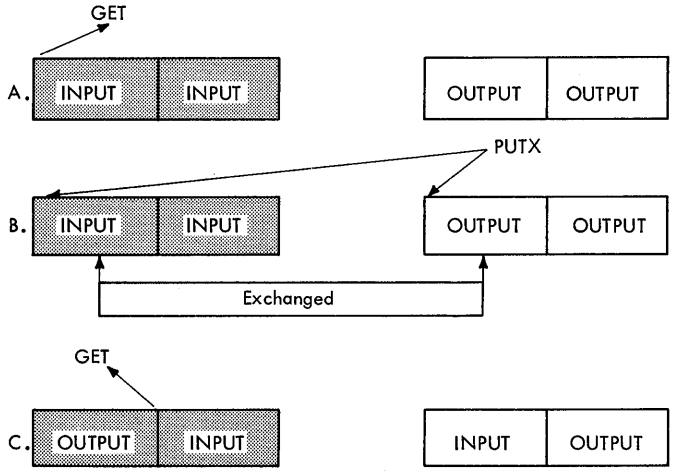



```
             LA   0,WORKAREA
NEXTREC      GET  INDCB,(0)
             LR   0,1
             ...
             PUT  OUTDCB,(0)
             LR   0,1
             B    NEXTREC
WORKAREA     DS   CL50
INDCB        DCB  MACRF=(GT),---
OUTDCB       DCB  MACRF=(PT),---
```

Figure 23. Exchange Buffering (GT, PT)

Exchange Buffering -- GET-locate, PUTX-output: The GET macro instruction (step A, Figure 24) locates the address of the next input record. The address is returned in register 1. The record must be processed in the buffer segment before the PUTX macro instruction (step B, Figure 24) is issued. The PUTX macro instruction specifies the address of both the input and output data control block. The two buffer segments are exchanged without any movement of data. The GET macro instruction (step C, Figure 24) locates the next record to be processed.

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




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

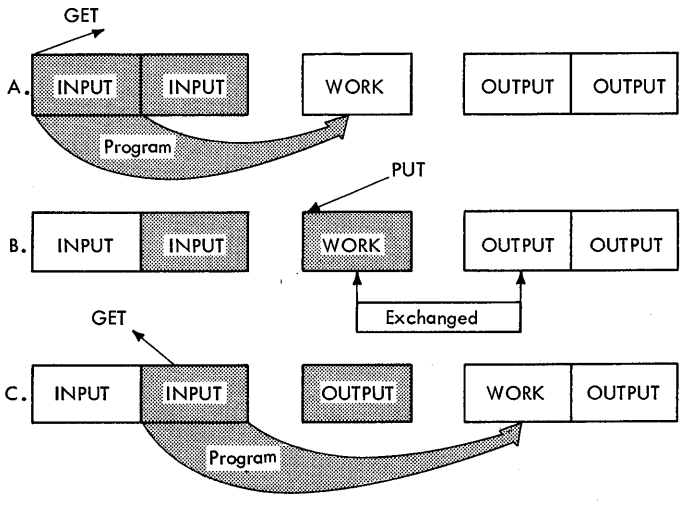
Figure 24. Exchange Buffering (GL, PM)

Exchange Buffering -- GET-locate, PUT-substitute: The GET macro instruction (step A, Figure 25) locates the next input record. Its address is returned in register 1. You must then move the record to a work area. You can process the record either before or after the move.

The PUT macro instruction (step B, Figure 25) specifies the address of the work area containing the next output record. The system returns the address of the next output buffer available for use as a work area in register 1. That address is passed to the move (MVC) instruction in register 6.

The GET macro instruction (step C, Figure 25) locates the next input record. You must then move the record to the new work area. Notice that the previous work area becomes part of the output buffer (step C).

Note: If records other than format F are being moved, the length attribute of the MVC instruction must be changed as shown. If the record is more than 256 bytes long, you must code a move routine to process the complete record.




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

Figure 25. Exchange Buffering (GL, PT)

Buffering Techniques and GET/PUT Processing Modes: As you can see from the previous examples, the most efficient coding is achieved by using automatic buffer pool construction, and GET-locate and PUTX-output with either simple or exchange buffering. Figure 26 summarizes the combinations of buffering techniques and processing modes that can be used. Notice, for example, that if you use PUT-locate and GET-substitute, you must provide a work area and you must also move the record from the work area to the output buffer.

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

[1]PUTX, only

Figure 26. Buffering Technique and GET/PUT Processing Modes

RELSE -- Release an Input Buffer

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

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

In an operating system with the time sharing option (TSO), you can issue a RELSE macro instruction from a program running in a time sharing

region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, the RELSE is treated as a NOP instruction.

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/PUTX-output mode.

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

The last block of a data set is truncated by the close routine.

Note: A data set containing format F records with truncated blocks cannot be read from direct access storage as efficiently as standard format F data sets.

In an operating system with the time sharing option (TSO), you can issue a TRUNC macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, the TRUNC is treated as a NOP instruction.

GETBUF -- Get a Buffer From a Pool

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

FREEBUF -- Return a Buffer to a Pool

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

FREEDBUF -- Return a Dynamic Buffer to a Pool

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

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

Processing a Sequential Data Set

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

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

DATA FORMAT -- DEVICE TYPE CONSIDERATIONS

Both the record format (RECFM) and device-dependent (DEVD) information must be provided to the operating system prior to execution of your program. This information can be supplied by 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 the Job Control Language Reference manual.

The record format (RECFM) parameter of the DCB macro instruction specifies the characteristics of the records in the data set as fixed length (F), variable-length (V or D), or undefined length (U). Fixed-length, blocked records (FB) can be specified as standard (FBS), that is, there are no truncated (short) blocks or unfilled tracks within the data set, with the possible exception of the last block or track. If the data set resides on a direct access volume, the track overflow feature (T) cannot be specified for the basic access technique.

If you plan to read a data set backwards or to extend it at a later time, proceed as follows when coding RECFM for the creation of a sequential data set:

- If you know you will not have any truncated blocks, you can specify RECFM=FBS.
- If you are uncertain about whether you will have a truncated block, you should specify RECFM=FB.

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 devices. 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 data control block is ignored. The optional control character must be in the first byte of format F and U records and in the fifth byte of format V records or format D records where BUFOFF=(L). Control character codes are listed in Appendix B.

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

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

MAGNETIC TAPE (TA)

Format F, V, D, and U records are acceptable for magnetic tape. Format V records are not acceptable on 7-track tape if the data conversion feature[1] is not available. ASCII records are not accepted 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. For format D records (ASCII), the padding consists of ASCII circumflex characters.

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

Tape density (DEN) specifies the recording density in bits per inch per track, as shown in Figure 27. If this information is not supplied, the highest applicable density is assumed.

DEN Value	Recording Density Model 2400			
	7-Track	9-Track	9-Track (phase encoded)	9-Track (dual density)
0	200	-	-	-
1	556	-	-	-
2	800	800	-	800[1]
3	-	-	1600	1600[2]

[1]Non-return-to-zero IBM (NRZI) mode
[2]Phase encoding (PE) mode

Figure 27. Tape Density (DEN) Values

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

C - data conversion is to be used.
E - even parity is to be used; if omitted, odd parity is assumed.
T - BCDIC to EBCDIC translation is required.

---

[1]Data conversion makes it possible to write eight binary bits of data on seven tracks. Otherwise, only six 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.

PAPER TAPE READER (PT)

The paper tape reader accepts format F or U records. Each format U record is followed by an end-of-record character. Data read from paper tape is optionally converted into the System/360 internal representation of one of six standard paper tape codes. Any character found to have a parity error will not be converted when the record is transferred into the input area. Characters deleted in the conversion process are not counted in determining the block size.

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

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

Note: When using QSAM, the processing mode must be move mode.

CARD READER AND PUNCH (RD/PC)

Format F, V, or U records are acceptable to both the reader and punch. The device control character, if specified in the RECFM parameter, is used to select the stacker; it is not punched. The first 4 bytes (record descriptor word or segment descriptor word) of format V records or record segments are not punched. For format V records, at least one 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) and 160 (column binary mode) data bytes. If mode (C) is used, the DCB parameters BLKSIZE, LRECL, and BUFL must be specified as 160. You can specify the read/punch mode of operation (MODE) as either card image (column binary) mode (C) or EBCDIC mode (E). If this information is omitted, E is assumed.

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

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

PRINTER (PR)

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

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

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

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

DIRECT ACCESS (DA)

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

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

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

SEQUENTIAL DATA SETS -- DEVICE CONTROL

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

When using 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. All read or write operations must be checked before issuing a device control macro instruction.

CNTRL -- Control an I/O Device

The CNTRL macro instruction provides a number of 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.

In an operating system with the time sharing option (TSO), you can issue a CNTRL macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, the CNTRL is treated as a NOP instruction.

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

PRTOV -- Test for Printer Overflow

The PRTOV macro instruction tests for channel 9 or 12 of the printer carriage control tape. An overflow condition will cause 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 data control block is not a printer, no action is taken.

In an operating system with the time sharing option (TSO), you can issue a PRTOV macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, the PRTOV is treated as a NOP instruction.

SETPRT -- Load Character Set for UCS Printer

The SETPRT macro instruction indicates the character set to be used by a 1403 printer with the Universal Character Set feature. It thus allows your program to change character sets during execution; as an option, it allows lower-case alphabetic characters to be printed in uppercase when no uppercase/lowercase print chain is available.

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

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

In an operating system with the time sharing option (TSO), you can issue a SETPRT macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB statement contains the operand TERM=TS, the SETPRT is treated as a NOP instruction.

BSP -- Backspace a Magnetic Tape or Direct Access Volume

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

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

In an operating system with the time sharing option (TSO), you can issue a BSP macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, the BSP is treated as a NOP instruction.

NOTE -- Return the Relative Address of a Block

The NOTE macro instruction requests the relative address of the block just read or written. The feedback identifies the block for subsequent repositioning of the volume.

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

In an operating system with the time sharing option (TSO), you can issue a NOTE macro instruction from a program running in a time sharing region. If you do and if the DD statement associated with the DCB contains the operand TERM=TS, the NOTE is treated as a NOP instruction.

POINT -- Position to a Block

The POINT macro instruction causes repositioning of a magnetic tape or direct access volume to a specified block in the data set. The next read or write operation begins at this block.

SEQUENTIAL DATA SETS -- TERMINAL CONTROL

The operating system with the time-sharing option (TSO) provides macro instructions that you can issue from a program running in a TSO foreground region, the region containing time-sharing jobs. Using these macro instructions lets you control several programming features related to the characteristics of the terminal and to the manipulation of buffers constructed by the TSO system. Because of the control functions of these macro instructions, you may want to use some of them near the beginning of your program. If you use them in an MVT batch entry (background) job, they are treated as NOP instructions. For the details you need to code the macro instructions, refer to the Supervisor and Data Management Macro Instructions publication.

STBREAK -- Set Break Feature

Use the STBREAK macro instruction to use or to suppress the reverse break (also called receive interrupt) feature, which is applicable only to the IBM 1050 and 2741 terminals. The reverse break feature lets output directed to the terminal interrupt input being entered at the terminal.

Issuing the STBREAK macro instruction with the YES operand allows you to specify the BREAKIN operand when you issue a TPUT macro instruction. The reverse break feature, used through the BREAKIN operand, allows the output from your program to interrupt the terminal user who is keying in data.

STCOM -- Set Status of Interterminal Communications

For any output operations to the terminal, such as the final printing of documents, your foreground job can stop messages coming from other terminals by issuing the STCOM macro instruction. If your program issues a TPUT macro instruction with the TJID operand after issuing STCOM to stop interterminal communications, register 15 will contain a return code of 0C when your program regains control.

If you issue STCOM in an operating system that does not include the time sharing option, STCOM is ignored.

STTIMEOU -- Specify Terminal Timeout Feature

Use the STTIMEOU macro instruction to use or to suppress the timeout feature on IBM 1050 terminals. The timeout feature locks the terminal keyboard after approximately 20 to 30 seconds of inactivity. The macro instruction is ignored if it is issued for terminals other than the IBM 1050.

If you issue STTIMEOU in an operating system that does not have the time sharing option, STTIMEOU is ignored.

STCC -- Specify Terminal Control Character

Use the STCC macro instruction to specify what control characters can be used by the terminal user for character and line deletion. The control characters can be any character on the terminal keyboard, with the exceptions noted in the Supervisor and Data Management Macro Instructions publication.

You can also use STCC to specify that when the terminal user depresses the attention key at the end of a line being entered, that line is deleted.

If you issue STCC in an operating system that does not include the time sharing option, STCC is ignored.

STATTN -- Set Attention Simulation

The terminal user can interrupt instruction execution by pressing the terminal attention key. For terminals without an attention key, use STATTN to specify conditions through which the terminal user can initiate an attention interrupt.

If you issue STATTN in an operating system that does not have the time sharing option, STATTN is ignored.

GTSIZE -- Get Terminal Line Size

In your foreground job, the GTSIZE macro instruction finds the terminal line size and returns it in register 1. If the terminal is a display station, GTSIZE also finds the screen length and returns it in register 0. You can test these values to ensure their suitability for any given operation and then make any necessary adjustment with the STSIZE macro instruction.

If you issue GTSIZE in an operating system that does not have the time sharing option, GTSIZE is ignored.

STSIZE -- Set Terminal Line Size

Use the STSIZE macro instruction to set the line size of the terminal. If the terminal is a display station, use STSIZE to set the screen size, too. If you set the terminal line size or the display station screen size to a value that is longer than the actual length of the terminal line or the screen, multiple lines are sent to the terminal or to the display screen. If you generate a record that is longer than the terminal line or the screen, multiple lines are sent to the terminal or to the display screen.

If you issue STSIZE in an operating system that does not have the time sharing option, STSIZE is ignored.

STAUTOLN -- Start Automatic Line Numbering

The STAUTOLN macro instruction causes line numbering to be generated by the system. When specified with certain input operations, an input data set can be line numbered as specified by STAUTOLN parameters.

When operating with the PROMPT option of the PROFILE terminal command, STAUTOLN can be used to have the system issue a numeric prompt message with each request for data input.

If you issue STAUTOLN in an operating system that does not have the time sharing option, STAUTOLN is ignored.

STAUTOCP -- Start Automatic Character Prompting

Use the STAUTOCP macro instruction to start automatic character prompting. Automatic character prompting is a terminal feature used to signal the terminal user when the system is ready to accept input from the terminal. Its signal consists of typing at the terminal either an underscore and a backspace or a period and a carriage return, depending on the type of terminal being used.

If you issue STAUTOCP in an operating system that does not have the time sharing option, STAUTOCP is ignored.

SPAUTOPT -- Stop Automatic Prompting

Use the SPAUTOPT macro instruction to terminate automatic line numbering and character prompting. After issuing this macro instruction, you can reinitiate character prompting and line numbering only by reissuing STAUTOCP and STAUTOLN.

If you issue SPAUTOPT in an operating system that does not have the time sharing option, SPAUTOPT is ignored.

RTAUTOPT -- Restart Automatic Prompting

The terminal user can stop either automatic character prompting or automatic line numbering by entering a null line (carriage return) or by striking the attention key. You can restart automatic character prompting or automatic line numbering at the point of interruption by issuing the RTAUTOPT macro instruction.

If you issue RTAUTOPT in an operating system that does not have the time sharing option, RTAUTOPT is ignored.

STCLEAR -- Set Character String to Clear Display Station Screen

Use the STCLEAR macro instruction to specify the character string that the terminal user can enter to erase a display station screen.

If you issue STCLEAR in an operating system that does not have the time sharing option, STCLEAR is ignored.

TCLEARQ -- Clear TSO Buffers

The TCLEARQ macro instruction clears either the input or the output buffers that TSO uses to pass data between a foreground job and a terminal device.

Use this macro instruction when your foreground job and data sent from the terminal user are not synchronized. This can occur when your foreground job requests input from the terminal and receives input of a different type or format than that requested. Your foreground job must then clear the TSO input buffers that may contain additional terminal

input and request that the unacceptable data be reissued in the correct type or format. The keyboard is locked until your job issues the next TGET macro instruction.

Also use the TCLEARQ macro instruction when you encounter an error situation and wish to issue a message. To do this immediately, clear the output buffers with a TCLEARQ macro instruction and then send your messages to the terminal.

If you issue TCLEARQ in an operating system that does not have the time sharing option, TCLEARQ is ignored.

SEQUENTIAL DATA SETS -- DEVICE INDEPENDENCE

Device independence is an important consideration when programming the System/360 or System/370. The ability to request input/output operations without regard for the physical characteristics of the I/O devices makes it possible for you to write one program that will fulfill a variety of needs. Device independence may be useful for:

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

Device independence is clearly a valuable concept -- one that is not difficult to achieve, but which requires some planning and forethought. There are two areas of planning necessary to achieve device independence -- system generation considerations and programming considerations.

SYSTEM GENERATION CONSIDERATIONS

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

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

Effecting a smooth transition to new input/output devices must not be construed to mean the inclusion of unsupported devices. This discussion is limited to add-on or substitution device independence. When support for new devices is provided, a new system will have to be generated. A

complete description of system generation techniques is contained in the System Generation manual.

PROGRAMMING CONSIDERATIONS

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

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

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

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

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

READ
: specify forward reading only (SF).

WRITE
: specify forward writing only (SF); use only to create new records.

PUTX
: use only output mode.

NOTE/POINT
: valid for both magnetic tape and direct access volumes.

BSP
: valid for magnetic tape or direct access volumes. However, its use would be an attempt to perform device-dependent action.

CNTRL/PRTOV
: 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 is apt to be used. Magnetic tape and unit record equipment data control blocks will fit in the area provided during assembly. Specify unit record devices only if you expect never to change to tape or direct access devices. Key length (KEYLEN) can be specified on the DD statement if necessary.

RECFM, LRECL, BLKSIZE
these can be specified in the DD statement. However, you must consider maximum record size for specific devices. Also, track overflow cannot be specified unless supported.

DSORG
specify sequential (PS/PSU).

OPTCD
device dependent; specify in the DD statement.

SYNAD
any device-dependent error checking is automatic. Generalize your routine so that no device-dependent information is required.

CHAINED SCHEDULING FOR I/O OPERATIONS

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

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

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

- BDAM CREATE, that is, MACRF=(WL).
- Track overflow.
- Operand of the OPEN macro instruction specifies UPDAT.
- Exchange buffering.
- CNTRL macro instruction to be used.
- Device type is paper tape reader.

When chained scheduling is being used, the automatic skip feature of the PRTOV macro instruction for the printer will not function. Format control must be achieved by ANSI or machine control characters. (Control characters are discussed in more detail in Part 1 under "Control Character," in Part 2 under "Data Format -- Device Type Considerations," and in Appendix B.) When you use undefined length records with QSAM, the DCBLRECL field represents the block size instead of the actual record length.

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

CREATING A SEQUENTIAL DATA SET

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

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

Tape-to-Print, Move Mode -- Simple Buffering: In Example 6, the GET-move and PUT-move require two movements of the data records. If the record length (LRECL) does not change in processing, only one move is necessary; you can process the record in the input buffer segment. A GET locate can be used to provide a pointer to the current segment.

```
          ...
          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 mess-
          LA        0,68(0,1)                age address in register 1.
          ST        14,SAVE14                SYNAD routine prints part of
          PUT       OUTDATA,(0)              the message (beginning with
          SYNADRLS                           the unit number) as a 56-byte
          L         14,SAVE14                fixed-length record. It then
          RETURN                             returns to the control
ENDJOB    CLOSE     (INDATA,,OUTDATA)        program.
          ...
COUNT     DS        CL6
WORKAREA  DS        CL50
NUMBER    DC        PL4'0'
SAVE14    DS        F
INDATA    DCB       DDNAME=INPUTDD,DSORG=PS,MACRF=(GM),EROPT=ACC,       C
                SYNAD=TAPERROR,EODAD=ENDJOB
OUTDATA   DCB       DDNAME=OUTPUTDD,DSORG=PS,MACRF=(PM),EROPT=ACC
```

Example 6. Creating a Sequential Data Set -- Move Mode, Simple Buffering

Tape-to-Print, Locate Mode -- Simple Buffering: This problem (Example 7) is similar to Example 6. However, since there is no change in the record length, the records can be processed in the input buffer. Only one move of each data record is required.

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

Example 7. Creating a Sequential Data Set -- Locate Mode, Simple Buffering

Tape-to-Print, Substitute Mode -- Exchange Buffering: Although the initial problem is the same, the solution described in Example 8 takes advantage of two facilities: exchange buffering, which eliminates theneed to move the data record; and direct reference to individual fields within a record through the use of a dummy control section (DSECT). The use of the DSECT allows symbolic reference to be made for storage-to-storage operations; therefore, the length attributes need not be explicitly stated.

```
          ...
          OPEN      (INDATA,,OUTDATA,(OUTPUT),ERRORDCB,(OUTPUT))
          LA        0,GIVEAWAY                Set up for first buffer
NEXTREC   GET       INDATA,(0)                Substitute Mode
          LR        2,1                       Pointer to next record
          USING     RECORD,2                  Establish address of DSECT
          AP        NUMBER,=P'1'
          UNPK      COUNT,NUMBER
          PUT       OUTDATA,RECORD            Substitute Mode
          LR        0,1                       Exchange work area
          B         NEXTREC
TAPERROR  SYNADAF   ACSMETH=QSAM              SYNAD routine is same
          ...                                 as in previous example
ENDJOB    CLOSE     (INDATA,,OUTDATA,,ERRORDCB)
          ...
          DS        0D
GIVEAWAY  DS        CL50
NUMBER    DC        PL4'0'
INDATA    DCB       DDNAME=INPUTDD,DSORG=PS,MACRF=(GT),BFTEK=E,BFALN=D, C
               EROPT=ACC,SYNAD=TAPERROR,EODAD=ENDJOB
OUTDATA   DCB       DDNAME=OUTPUTDD,DSORG=PS,MACRF=(PT),BFTEK=E,BFALN=D,C
               EROPT=ACC
          ...
RECORD    DSECT
COUNT     DS    ZL6
RESTOFIT  DS    CL44
```

Example 8. Creating a Sequential Data Set -- Substitute Mode, Exchange Buffering

USING BSAM TO READ FIXED BLOCKED RECORDS

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

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

  Obtain the residual count from the CSW that has been stored in the IOB. The residual count is in the halfword at decimal offset 14 from the start of the IOB. Subtract this residual count from the number of data bytes requested to be read by the READ macro instruction. If "S" was coded as the length parameter of the READ macro instruction or the length parameter was omitted, the number of bytes requested is the value of DCBBLKSI at the time the READ was issued. If the length was coded in the READ macro instruction, this value is the number of data bytes and it is contained in the halfword at offset 6 from the beginning of the DECB. The result of the subtraction is the length of the block read. See Example 9.

```
         OPEN      (DCB,(INPUT))
         LA        DCBR,DCB
         USING     IHADCB,DCB2
         ...
         READ      DECB1,SF,DCB,AREA1
         READ      DECB2,SF,DCB,AREA2,50
         ...
         CHECK     DECB1
         LH        WORK1,DCBBLKSI          Blocksize at time of READ
         L         WORK2,DECB1+16          IOB Address
         SH        WORK1,14(WORK2)         WORK1 now has block length
         ...
         CHECK     DECB2
         LH        WORK1,DECB2+6           Length requested
         L         WORK2,DECB2+16          IOB Address
         SH        WORK1,14(WORK2)         WORK1 has block length
         ...
         MVC       DCBBLKSI,LENGTH3        Length to be read
         READ      DECB3,SF,DCB,AREA3
         ...
         CHECK     DECB3
         LH        WORK1,LENGTH3           Blocksize at time of READ
         L         WORK2,DECB+16           IOB Address
         SH        WORK1,14(WORK2)         WORK1 has block length
         ...
DCB      DCB       ...RECFM=U,NCP=2,...
         DCBD
```

Example 9. Using BSAM to Read Fixed Blocked Records

Processing a Partitioned Data Set

A partitioned data set is divided into sequentially organized members made up of one or more records (see Figure 24). Each member has a unique name, one to eight characters long, stored in a directory. The records of a given member are stored or retrieved sequentially.

The main advantage of using a partitioned data set is that you can retrieve any individual member once the data set is opened. For example, a program library can be stored as a partitioned data set, each member of which is a separate program or subroutine. The individual members can be added or deleted as required. When a member is deleted, only the member name is removed from the directory; the space used by the member cannot be reused until the data set is reorganized.

The directory, a series of records at the beginning of the data set, contains an entry for each member. Each directory entry contains the member name and the starting location of the member within the data set, as shown in Figure 28. The directory entries are arranged in alphameric collating sequence by name. In addition, you can specify up to 62 characters of information in the entry.

The track address of each member is recorded by the system as a relative track within the data set rather than as an absolute track address. Thus, an entire data set can be moved without changing the relative track addresses. The data set can be considered as one continuous set of data tracks regardless of how the space was actually allocated. If there is not sufficient space available in the directory for an additional entry, or not enough space available within the data set for an additional member, no new members can be stored.

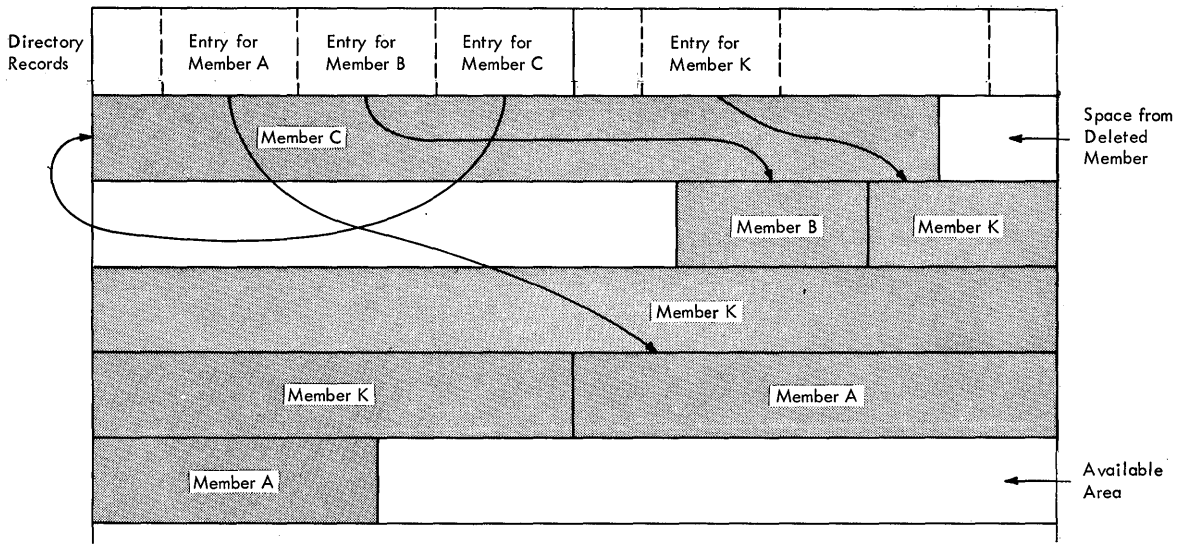


Figure 28. A Partitioned Data Set

PARTITIONED DATA SET DIRECTORY

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

Member entries vary in length and are blocked into 256-byte blocks. Each block contains as many complete entries as will fit in a maximum of 254 bytes; any remaining bytes are left unused and are ignored. Each directory block contains a 2-byte count field that specifies the number of active bytes in a block. As shown in Figure 29, 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.

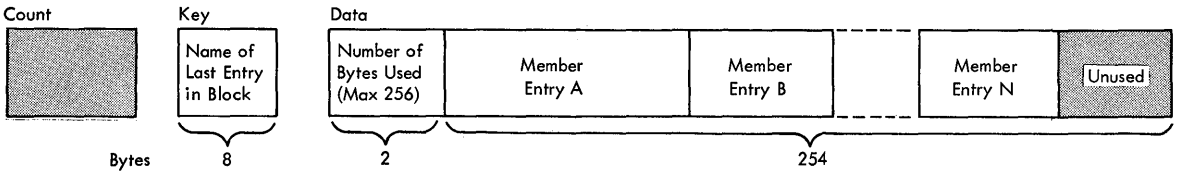


Figure 29. A Partitioned Data Set Directory Block

Each member entry contains a member name or alias; there can be up to 16 aliases (alternate names) for each member. Each entry also contains the relative track address of the member and a count field, as shown in Figure 30. 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 ones.

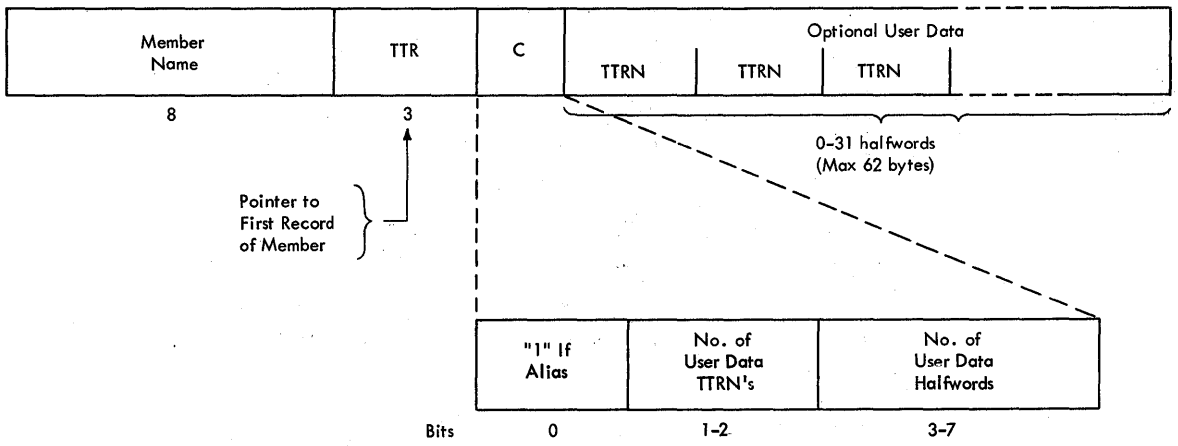


Figure 30. A Partitioned Data Set Directory Entry

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

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

Note: This pointer is created by adding one 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 one 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:

Bits 0 1-2 3-7

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

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

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

- More than three pointers are used in the user data field.
- The pointers in the user data field or note list do not conform to the standard format.

- The pointers are not placed first in the user data field.
- Any direct access addresses (absolute or relative) are embedded in any data blocks or in another data set that refers to this data set.

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

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

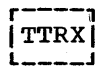
User Data

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. 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 the IEHMOVE utility program. Each 4-byte entry in the note list has the following format:



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

R is the relative block number on that track.

X is available for any use.

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

PROCESSING A MEMBER OF A PARTITIONED DATA SET

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

In order to locate a member or to process the directory, several macro instructions are provided by the operating system. The BLDL macro instruction can be used to structure a list of directory entries in main storage; the FIND macro instruction locates a member of the data set for subsequent processing; the STOW macro instruction adds or deletes a

member name in the directory. To use these macro instructions, you must specify DSORG=PO or POU in the DCB macro instruction. Before issuing a FIND, BLDL, or STOW macro instruction, you must check all preceding input/output operations for completion.

BLDL -- Construct a Directory Entry List

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

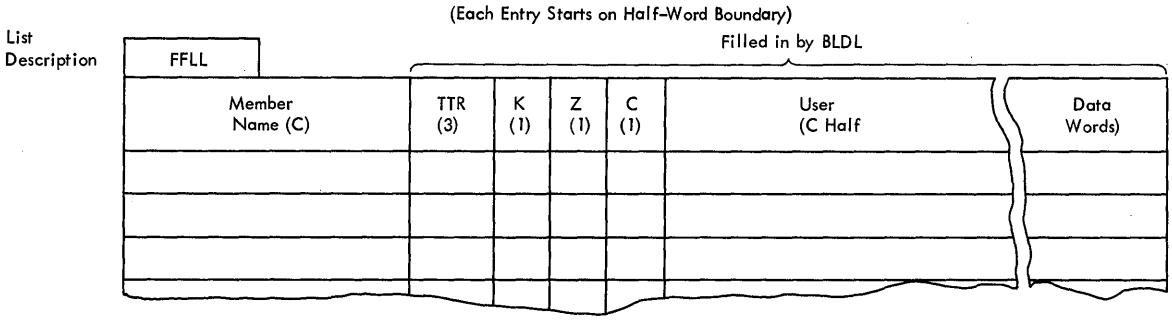
You can optimize retrieval time by directing a subsequent FIND macro instruction to the build list rather than the directory to locate the member to be processed.

The build list, as shown in Figure 31, 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 eight bytes of each entry contain the member name or alias. The next six bytes must be available to contain the starting address of the member plus some control data. If additional information is to be supplied from the directory, up to 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 instruction. If you want to find only one member, the function is performed automatically when you specify the data set name and member name in the related DD statement. The system places the correct address in the data control block so that a subsequent GET/READ macro instruction will begin processing at that point.

There are two ways in which the system can be directed to the desired member: you can specify the address of either an area containing the name of the member or an entry in a build list you have created. In the first case, the system searches the directory of the data set. If a build list is used, no search is required; the relative track address is determined from the list entry.



Programmer Supplies:
FF = Number of member entries in list
LL = Even no. giving byte length of each entry (minimum of 12)
Member name = eight bytes, left-adjusted

BLDL Supplies:
TTR = Member starting location
K = If only data set = 0
If concatenation = no.
Z = Normally padding for boundary alignment
C = Same C field from directory. Gives no. of user data halfwords
User data: as much as will fit in entry

Figure 31. Build List Format

STOW -- Alter a Directory Entry

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

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

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

CREATING A PARTITIONED DATA SET

If you have no need to add entries to the directory, i.e., the STOW and BLDL macro instructions will not be used, you can create a new data

set and write the first member as follows (see Example 10):

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

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

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.

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

Example 10. Creating One Member of a Partitioned Data Set

To take full advantage of the STOW macro instruction, and thus the BLDL and FIND macro instructions in future processing, you can provide additional information with each directory entry. This is accomplished by using the basic access technique, which also allows you to process more than one member without closing and reopening the data set, as follows (see Example 11):

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

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

```
OUTDCB    DCB   --,DSORG=PO,DDNAME=PDSDD,--
          OPEN  (OUTDCB,(OUTPUT))
          WRITE   *
          CHECK   First record of member.
          NOTE
          WRITE
          CHECK   Remaining records of member.
          (NOTE)  Only NOTE first record of a subgroup within member.
          WRITE
          CHECK   Write note lists at end of each
          NOTE    subgroup.
          STOW    Member entry in directory after all records and note
                  lists are written.

Repeat from * for each additional member

          CLOSE (OUTDCB)
```

Example 11. Creating Members of a Partitioned Data Set, Using STOW

RETRIEVING A MEMBER

To retrieve a specific member from a partitioned data set, either the basic or queued access technique can be used as follows (see Example 12):

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

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

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

```
INDCB     DCB    --,DSORG=PS,DDNAME=PDSDD,--
          OPEN   (INDCB)      Automatic Find
          GET (or READ)
          CLOSE  (INDCB)
```

Example 12. Retrieving One Member of a Partitioned Data Set

In order to process several members without closing and reopening, or to take advantage of additional data in the directory, the following technique should be used (see Example 13):

- Code DSORG=PO or POU in the DCB macro instruction.
- Build a list (BLDL) of needed member entries from the directory.

- Indicate in the DD statement the data set name of the partitioned data set, that is, DSNAME=name and DISP=OLD.
- Use the FIND or POINT macro instruction to prepare for reading the member records.
- The records may be read from the beginning of the member, or a note list may be read first, to obtain additional locations that POINT to sub-categories within the member.
- READ (and CHECK) the records until all those required have been processed.
- POINT to additional categories, if required, and READ the records.
- Repeat this procedure for each member to be retrieved.

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

```
INDCB    DCB    --,DSORG=PO,DDNAME=PDSDD,--
         OPEN   (INDCB)
         BLDL        Build a list of selected member names in main
                     storage.
         FIND (or POINT)
         READ        *Read note list.
         CHECK
         POINT       Locate subgroup by using note list.
         READ
         CHECK       Read member records

Repeat from * for each additional member.

         CLOSE  (INDCB)
```

Example 13. Retrieving Several Members of a Partitioned Data Set Using BLDL, FIND, and POINT

UPDATING A MEMBER

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

UPDATING IN PLACE

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

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

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

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

In Example 14, overlap is achieved by having a read or write request outstanding while each record is being processed. Note the use of execute- and list-form macro instructions, 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 (R3)
          WRITE  (2),SF,MF=E            Write updated R2
          (If R3 requires an update, branch to R3UPDATE)
          CHECK  (2)                    If R3 requires no update, check
          B      READRECD                 write for R2 and loop
R3UPDATE  CALL   UPDATE,((3))           Call routine to update R3
          WRITE  (3),SF,MF=E            Write updated R3
          CHECK  (2)                    Check write for R2
          CHECK  (3)                    Check write for R3
          B      READRECD               Loop
FINISH    CLOSE  (UPDATDCB)             End-of-data exit routine
          ...
```

Example 14. 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. Since space is allocated when the data set is created, there is no need to request additional space. Note, however, that a partitioned data set must be contained on one volume. If

sufficient space has not been allocated, the data set must be reorganized by the IEBCOPY utility program.

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

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

Processing an Indexed Sequential Data Set

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; or 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. This flexibility is possible due to the inherent organization of the data set.

Although the queued and basic access techniques can be used to process an indexed sequential data set, each has separate and distinct functions. The queued access technique must be used to create the 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. It can also be used to update the data set directly.

INDEXED SEQUENTIAL DATA SET ORGANIZATION

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

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

- Prime Area -- This area, also called the prime data area, contains data records and related track indexes. It exists for all ISAM data sets.
- Overflow Area -- This area contains 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 ISAM data set are analogous to the index card file in a library. For example, if the library user knows the name of the book or the author, he can look in the index card file and obtain a catalog number that will enable him to locate the book in the book files. He would then go to the shelves and proceed through each row until he found the shelf containing the book. Usually each row contains a sign to indicate the beginning and ending numbers of all books in that particular row. Thus, as he proceeded through the rows, he would compare the catalog number obtained from the index with the numbers posted on each row. Upon locating the proper row, he would then search

that row for the shelf that contained the book. Then he would look at the individual book numbers on that shelf until he found the particular book.

ISAM uses the indexes in much the same way to locate records in an indexed sequential data set. The operating system provides both the queued and basic access techniques to process an indexed sequential data set. The queued access technique is used to create the data set and add records to the end. It can also be used to sequentially process or update the records. The basic access technique is used to read or update records and to insert new records at any place in the data set.

As the records are written in what is referred to as the prime data 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 32.

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

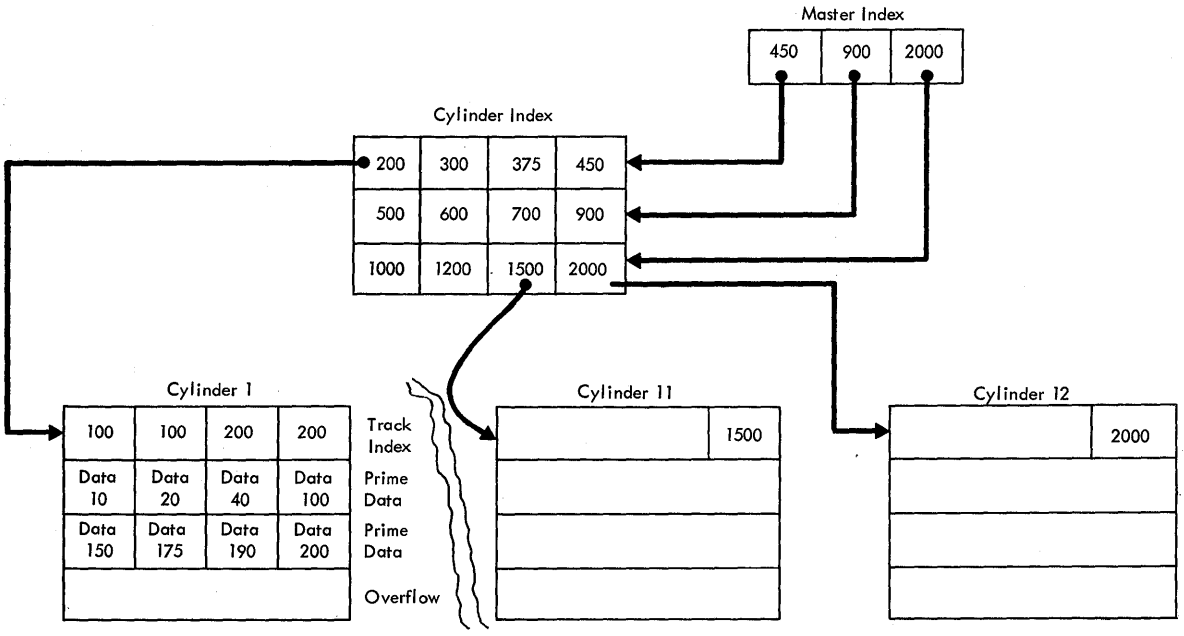


Figure 32. Indexed Sequential Data Set Organization

PRIME AREA

Records are written in the prime area when the data set is created or updated. The portion of Figure 33 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 preceding each block.

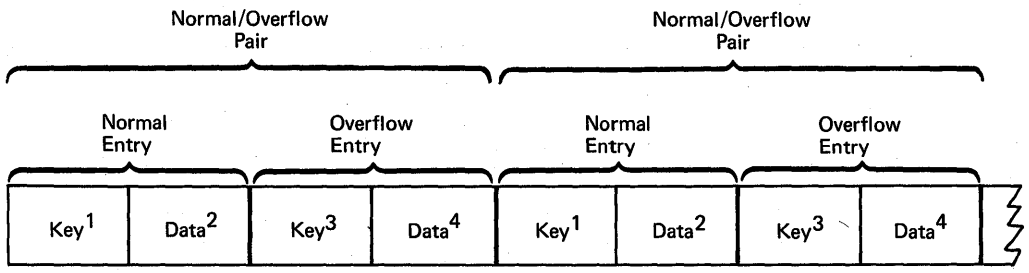
When the ABSTR option of the SPACE parameter of the DD statement is used to generate a multivolume prime area, the VTOC on the second volume and on all succeeding volumes must be contained within cylinder zero of the volume.

INDEX AREAS

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

Track Index: This is the lowest level of index and is always present. There is one track index for each cylinder in the prime area; it is written on the first tracks 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 zero or the first prime record on a shared track. For variable-length records, the normal entry contains the key of the highest record on the track and the home address of the prime 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 32.) 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 particular prime track. Figure 33 shows the format of a track index.



[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 33. Format of Track Index Entries

If all the tracks allocated for the prime data area are not used, their entries in the index are "flagged" as inactive. The last entry of each track index is a dummy entry indicating the end of the index. When fixed-length record format has been specified, the remainder of the last track used for a track index contains prime data records if there is room for them.

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

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

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

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

OVERFLOW AREAS

As records are added to an indexed sequential data set, space is required to contain those records that will not fit on the prime data track on which they belong. You can request that a number of tracks be set aside as a cylinder overflow area to contain overflows from prime tracks in each cylinder. An advantage of using cylinder overflow areas is a reduction of search time required to locate overflow records. A disadvantage is that there will be unused space if the additions are unevenly distributed throughout the data set.

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

If you request both cylinder overflow and independent overflow, cylinder overflow is used first.

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

ADDING RECORDS TO AN INDEXED SEQUENTIAL DATA SET

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

(key new); or they may be added to the prime data area by the queued access technique using the PUT macro instruction.

INSERTING NEW RECORDS INTO AN EXISTING INDEXED SEQUENTIAL DATA SET

As you add records to an indexed sequential data set, the system inserts each record in its proper sequence according to the record key. The remaining records on the track are then moved up one position. 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 "bumped" record to connect it logically to the correct track. The proper adjustments are made to the track index entries. This procedure is illustrated in Figure 34.

Subsequent additions are written either on the prime track where they belong 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.

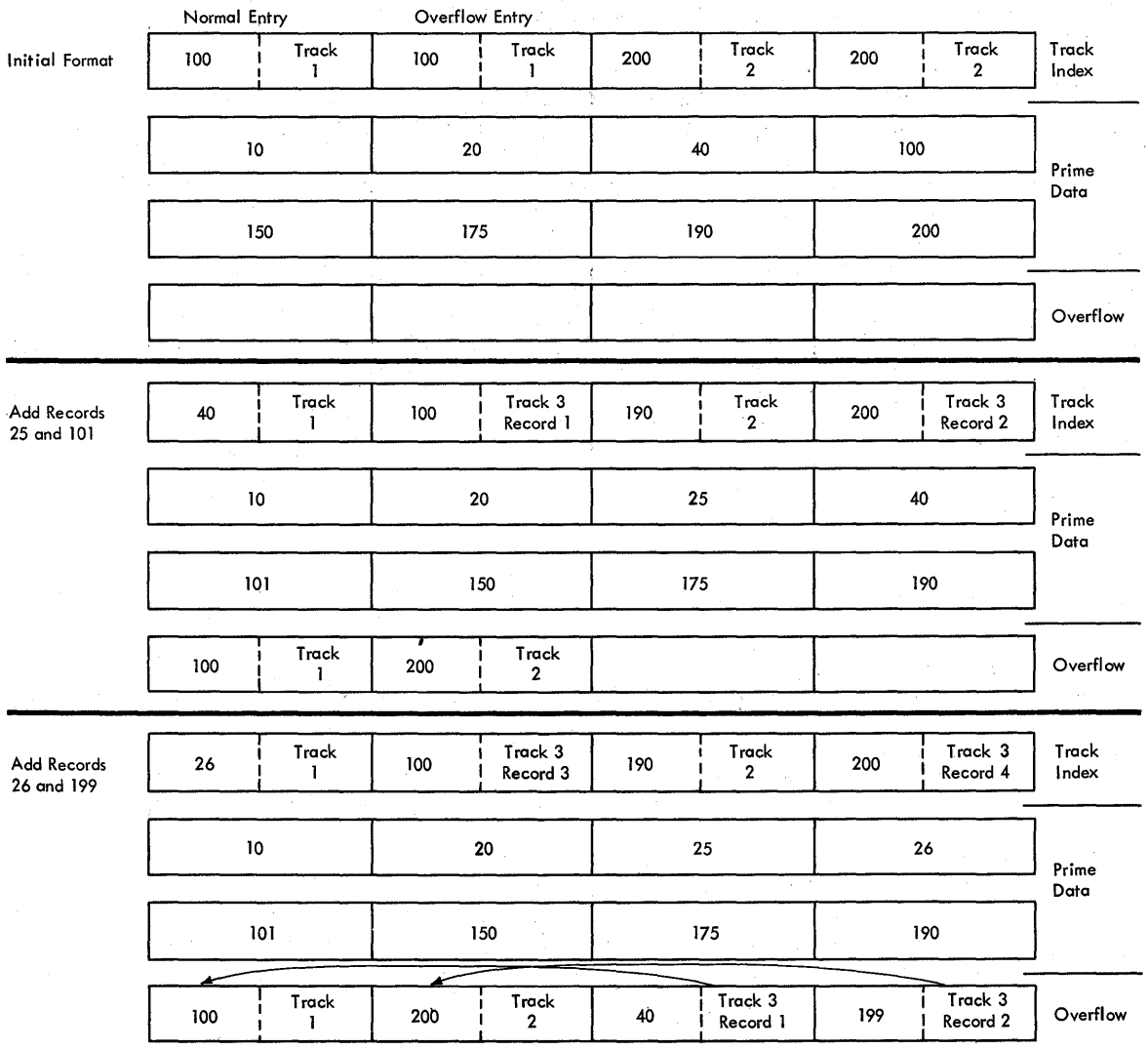


Figure 34. Adding Records to an Indexed Sequential Data Set

ADDING NEW RECORDS TO THE END OF AN INDEXED SEQUENTIAL DATA SET

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

When you use the WRITE KN macro instruction, the record being added will be 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 via the overflow chain even though additional prime data tracks originally allocated have not been filled.

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

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

- The key of any record to be added must be higher than the highest key currently in the data set.
- The DD statement must specify DISP=MOD.
- The data set must have been successfully closed when it was created or when records were previously added using the PUT macro instruction.

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

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

MAINTAINING AN INDEXED SEQUENTIAL DATA SET

An indexed sequential data set must be reorganized periodically 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 reorganizations:

- The data set can be reorganized in two passes by writing it sequentially into another area of direct access storage or magnetic tape and then recreating it in the original area.

- The data set can be reorganized 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 are written on the direct access volume and are available in the DCB for checking. The information includes 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 and appears in the RORG1, RORG2, and RORG3 fields of the DCB.

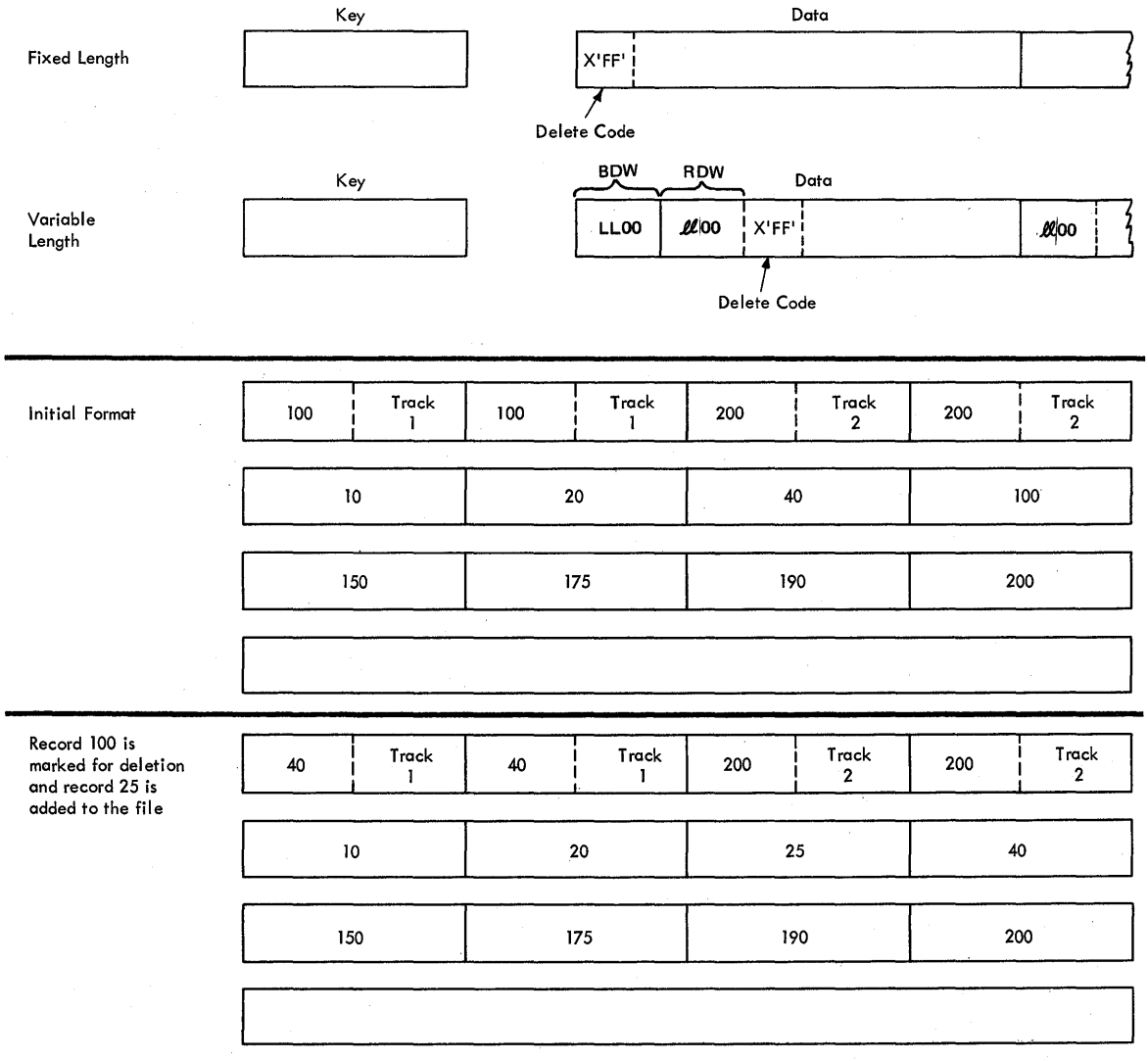


Figure 35. Deleting Records From an Indexed Sequential Data Set

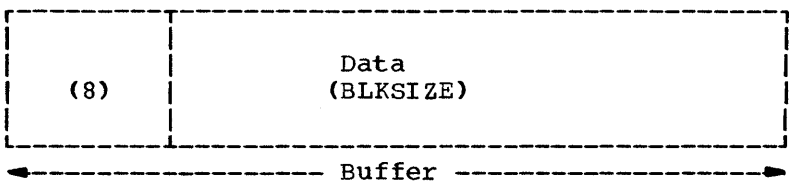
If you indicate when creating or updating the data set that you want to be able to flag records for deletion during updating, you can set the delete code to all ones (X'FF'). The delete code is the first byte of a fixed-length record or the fifth byte of a variable-length record. If a flagged record is forced off its prime track during a subsequent update, it will not be rewritten in the overflow area, as shown in Figure 35. Similarly, when you process sequentially, flagged records are not retrieved for processing. During direct processing, flagged records are retrieved like any other record and should be checked by you for the delete code.

Note: To use the delete option, RKP must be greater than 0 for fixed-length records and greater than 4 for variable-length records.

INDEXED SEQUENTIAL BUFFER AND WORK AREA REQUIREMENTS

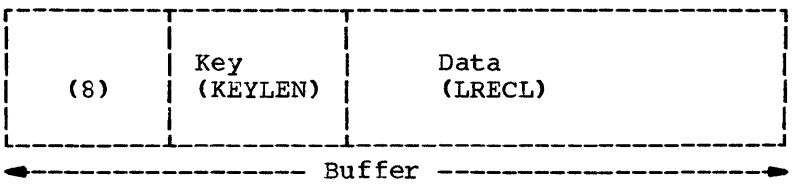
The only reason you will ever have to compute the buffer length (BUFL) requirements for your program is if you use the BUILD or GETPOOL macro instruction to construct the buffer area. If you are creating an indexed sequential data set (PUT macro instruction), each buffer must be eight bytes longer than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

Buffer length = 8 + Block size



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

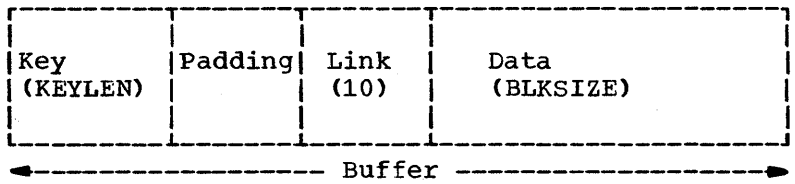
Buffer length = 8 + Key length + Record length



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

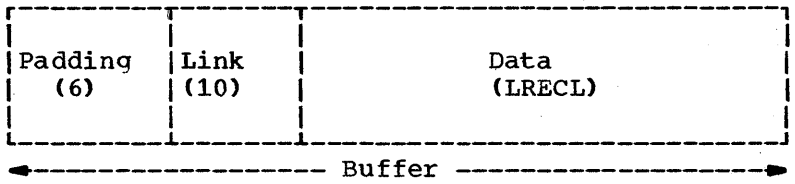
For fixed-length, unblocked records when both the key and data are to be read and for variable-length, unblocked records, padding is added so that the data will be on a doubleword boundary, that is:

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



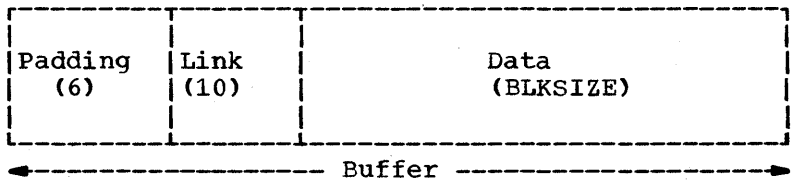
For fixed-length, unblocked records when only data is to be read:

Buffer length = 16 + LRECL



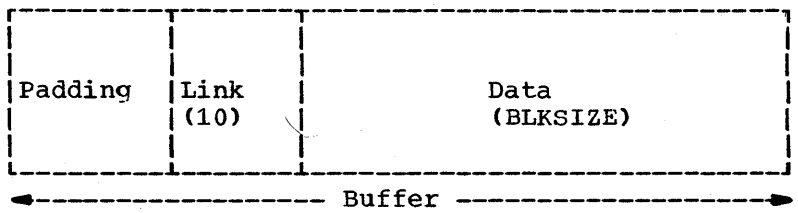
For fixed-length, blocked records:

Buffer length = 16 + BLKSIZE



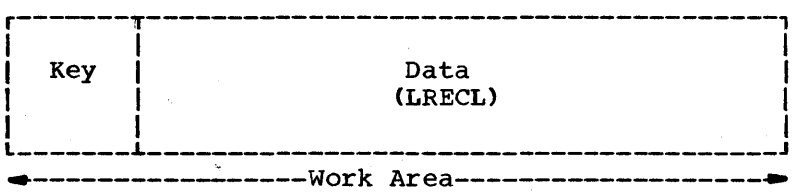
For variable-length, blocked records, padding is 2 if the buffer starts on a fullword boundary that is not also a doubleword boundary or it is 6 if the buffer starts on a doubleword boundary, that is:

Buffer length = 12 or 16 + Block size



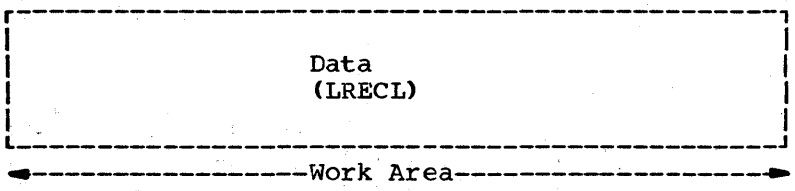
If you are using the input data set with fixed-length, unblocked records as a basis for creating a new data set, a work area is required. The size of the work area is given by:

Work area = Key length + Record length



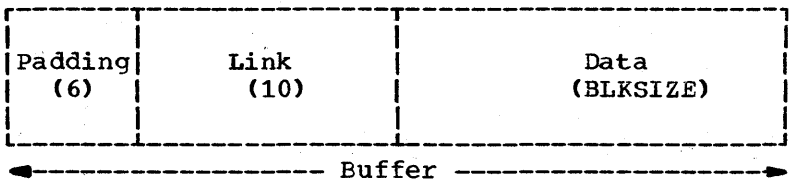
If you are reading only the data portion of fixed-length unblocked records or variable-length records, the work area is the same size as the record length, that is:

Work area = Record length



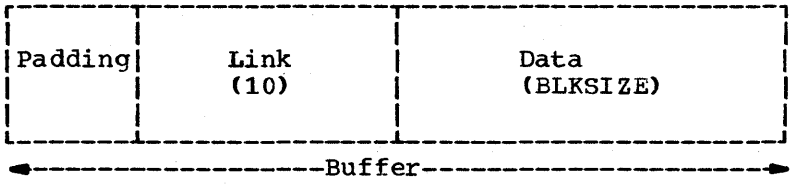
When using the basic access technique to update records in an indexed sequential data set, the key length field need not be considered in determining your buffer requirements. The area for fixed-length records must be:

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 it is 6 if the buffer starts on a doubleword boundary. Thus, the area must be:

Buffer length = 12 or 16 + Block size



When adding variable-length records to a data set, you may provide a special work area for the operating system by using the MSWA parameter of the DCB macro instruction. Although the work area is not required when you add fixed-length records, insertion is considerably expedited

if you provide such an area. 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 (SMSW parameter) varies according to the record format and the device type. You can calculate it during execution using device dependent information obtained with the DEVTYPE macro instruction and data set information from the data set control block (DSCB) obtained with the OBTAIN macro instruction. The DEVTYPE and OBTAIN macro instructions are discussed in the System Programmer's Guide.

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

For calculating the size of the work area, refer to the storage device capacities shown in Figure 36 under "Estimating Space Requirements" and the device overhead formulas given in the same section.

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

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

where Bn is the length of the last block on the track and Bi is the length of any block but the last as given in Figure 37 in the section "Estimating Space Requirements".

For variable-length records, SMSW may be calculated by one of two methods. The first method may lead to faster processing although it may require more main storage than the second method. For either method you must determine the value for HIRPD from the format 2 data set control block (DSCB). For the specific location of DS2HIRPD field in the data set control block, refer to the System Control Blocks publication.

The first method is as follows:

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

The second method is as follows;

$$SMSW = \left(\frac{\text{Track Capacity-Bn}}{\text{Bi}} + 2\right)(BLKSIZE) + 8(HIRPD) + LRECL + KEYLEN + 10$$

In all of the above formulas, the terms BLKSIZE, LRECL, KEYLEN, and SMSW are the same as the parameters in the DCB macro instruction. The second method yields a minimum value for SMSW. Therefore, the first method is valid only if its application results in a value higher than the value that would be derived from the second method. If neither MSWA nor SMSW is specified, the control program supplies the work area for variable-length records, using the second method to calculate the size.

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

the volume. The size of the storage area (SMSI parameter) varies. To allocate that space during execution, you can find the size of the index in the data control block (DCBNCRHI field) after the data set is opened or in the format 2 data set control block field, DS2NOBYT. Using the procedure discussed under the DCBD macro instruction, the storage area can be allocated and the SMSI field completed.

CONTROLLING AN INDEXED SEQUENTIAL DATA SET DEVICE

Processing of an indexed sequential data set is generally done in one of two ways: sequentially or directly. Direct processing is accomplished by using the basic access technique. Because you provide the key for the record you want read or written, all device control is handled automatically by the system. If you are processing the data set sequentially, using the queued access technique, the device is automatically positioned at the beginning of the data set.

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

SETL -- Specify Start of Sequential Retrieval

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

Use of a key prefix is extremely useful because you do not have to know the whole key of the first record to be processed. Any number of key characters can be used in the key prefix. Key characters to the right should be represented by binary zeros.

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

ESETL -- End Sequential Retrieval

The ESETL macro instruction directs the system to stop retrieving records from an indexed sequential data set. A new scan limit can then be set, or processing terminated. An end-of-data-set indication automatically terminates retrieval.

CREATING AN INDEXED SEQUENTIAL DATA SET

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

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

- Code DSORG=IS or ISU and MACRF=PM or PL in the DCB macro instruction.
- Specify in the DD statement the DCB attributes DSORG=IS or ISU, record length (LRECL), block size (BLKSIZE), record format (RECFM), key length (KEYLEN), relative key position (RKP), options required (OPTCD), cylinder overflow (CYLOFL), and the number of tracks for a master index (NTM). Specify space requirements with the SPACE parameter. To reuse previously allocated space, omit the SPACE parameter and code DISP=(OLD,[KEEP]).
- Open the data set for output.
- Use the PUT macro instruction to place all the records or blocks on the direct access volume.
- Close the data set.

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

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

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

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

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

Tape-to-Disk -- Indexed Sequential Data Set: This problem (Example 15) requires the creation of an indexed sequential data set from an input tape containing 60-character records. The key by which the data set is organized is in positions 20-29. The output records will be an exact image of the input, except that the records will be blocked. One track per cylinder is to be reserved for cylinder overflow. Master indexes are to be built when the cylinder index exceeds six tracks. Reorganization information about the status of the cylinder overflow areas is to be maintained by the system. The delete option will be used during any future updating.

```
//INDEXDD  DD     DSNAME=SLATE.DICT(PRIME),DCB=(BLKSIZE=240,CYLOFL=1,  C
//             DSORG=IS,OPTCD=MYLR,RECFM=FB,LRECL=60,NTM=6,RKP=19,     C
//             KEYLEN=10),UNIT=2311,SPACE=(CYL,25,,CONTIG),---
//INPUTDD  DD     ---
```

```
ISLOAD     START  0
           ...
           DCBD   DSORG=IS
ISLOAD     CSECT
           OPEN   (IPDATA,,ISDATA,(OUTPUT))
NEXTREC    GET    IPDATA            Locate Mode
           LR     0,1               Address of Record in Reg. 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
```

Example 15. 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 shorter 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 instruction to terminate processing. If no SYNAD routine is provided, the control program will terminate your processing. 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 only to add the new records, rather than to that necessary to recreate the whole data set.

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

During resume load on a data set with a partially filled track and/or a partially filled cylinder, the track index entry and/or the cylinder

index entry is overlaid when the track or cylinder is filled. If resume load abnormally terminates after these index entries have been overlaid, a subsequent resume load will 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 high key of the data set.

UPDATING AN INDEXED SEQUENTIAL DATA SET

In order to sequentially retrieve and update an indexed sequential data set:

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

Sequential Updates -- Indexed Sequential Data Set: Using the data set created in the previous example, you are to retrieve all records beginning with 915. Those records with a date (positions 13-16) previous to 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 code this problem is shown in Example 16.

```
//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 Reg. 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 DCBEXCD2 for error indication
*Error Routines
ENDJOB     CLOSE  (ISDATA)
           ...
ISDATA     DCB    DDNAME=INDEXDD,DSORG=IS,MACRF=(GL,SK,PU),          C
               SYNAD=CHECKERR
```

Example 16. Sequentially Updating an Indexed Sequential Data Set

DIRECT RETRIEVAL AND UPDATE OF AN INDEXED SEQUENTIAL DATA SET

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

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

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

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

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

To update existing records, it is recommended that you use the READ KU and WRITE K combination. However, if you use a WRITE K with a DECB not previously used to read the record, you are responsible for setting the overflow-record bit in the exception code field of the DECB. For blocked records, the overflow-record bit must be off when you write a prime block and on when you write an overflow block.

If there is a possibility that another program will require the use of the data set you are updating, you should ensure that you maintain exclusive control of at least the track. If you fail to maintain exclusive control of the data set that you are updating and if another data control block is opened before your data control block is closed, your updated records can become permanently inaccessible. In other words, when more than one data control block is open for updating a data set, the results are unpredictable. Exclusive control can be requested by using the ENQ macro instruction, which is described in the Supervisor Services book.

Direct Update With Exclusive Control -- Indexed Sequential Data Set: In this problem (Example 17) the previously described data set is to be updated directly with transaction records on tape. The input tape records are 30 characters long; the key is in positions 1-10; the update information is in positions 11-30. The update information replaces data in positions 31-50 of the indexed sequential data record.

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

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

Example 17. Directly Updating an Indexed Sequential Data Set

Note the use of the FREEDBUF macro instruction in Example 17. Usually the FREEDBUF macro instruction has two functions:

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

In Example 17, since the read operation was unsuccessful, the FREEDBUF macro instruction only frees the dynamically obtained buffer.

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

However, you can accomplish an automatic FREEDBUF simply by reusing the DECB, that is, by issuing another READ or a WRITE KN to the same

DECB. You should use this feature whenever possible, since it performs the functions of the FREEDBUF more efficiently. In Example 17, the FREEDBUF macro instruction should have been eliminated, since the WRITE KN addressed the same DECB as the READ KU.

For an indexed sequential data set with variable-length records, you may make three types of updates by using the basic access technique. You may read a record and write it back with no change in its length, simply updating some part of the record. This is done with a READ KU followed by a WRITE K, the same way you update fixed-length records. Two other methods for updating variable-length records use the WRITE KN macro instruction and allow you to change the record length. In one method, a record read for update (READ KU) may be updated in a manner that will change the record length and then be written back with its new length using WRITE KN. In the second method, you may replace a record with another record having the same key and possibly a different length using the WRITE KN macro instruction. To replace a record it is not necessary to have first read the record. In either method, when changing the record length, you must place the new length in the DECBLGTH field of the data event control block before issuing the WRITE KN macro instruction.

Direct Update -- Indexed Sequential Data Set with Variable-Length Records: In Example 18 an indexed sequential data set with variable-length records is updated directly with transaction records on tape. The transaction records are variable-length and each contains a code identifying the type of transaction. The transaction code 1 indicates that an existing record is to be replaced by one with the same key; 2 indicates that the record is to be updated by appending additional information, thus changing the record length; 3 or greater indicates that the record is to be updated with no change to its length. For this example, the maximum record length of both data sets is 256 bytes. The key is in positions 6-15 of the records in both data sets. The transaction code is in position 5 of records on the transaction tape. The work area (REPLAREA) 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 replace or other
           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 REPLAREA+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=(RUS,WUA),SYNAD=CHECKERR
TPDATA     DCB   ---
```

Example 18. Directly Updating an Indexed Sequential Data Set with Variable-Length Records

Processing a Direct Data Set

In a direct data set, there is a definite relationship between the 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. The actual organization of the data set is completely determined by you. If the data set has been carefully organized, location of a particular record takes less time than with an indexed sequential data set.

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

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

ORGANIZING A DIRECT DATA SET

In developing the organization of your data set, you can use a technique known as 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. For example, a data set with keys ranging from 0 to 4999 should be allocated space of 5000 records. Each key relates directly to a location that you can refer to as a relative record number. The main disadvantage of this type of organization is that records may not exist for many of the keys even though space has been reserved for them.

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

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

REFERRING TO A RECORD IN A DIRECT DATA SET

Once you have determined how your data set is to be organized, you must consider how the individual records will be referred to when the data set is updated or new records are added. This is important for determining whether feedback will be required when creating the data; if so, in what form the returned address will be used. The record identification can be represented in any of the forms described below.

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.

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.

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

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

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

To use the extended search option, you must indicate in the data control block 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, however, 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 data control block specifies a number equal to or greater than the number of tracks allocated to the data set or the number of records within the data set, the entire data set is searched in the attempt to satisfy your request.

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

CREATING A DIRECT DATA SET

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

If you are using a direct addressing technique with keys, you can reserve space for future records by writing a dummy record. A track for format U or V records can be reserved or truncated by writing a "capacity" record (see "Direct Access Device Characteristics").

Format F records are written sequentially as they are presented. When a track is filled, the system automatically writes the capacity record and advances to the next track. Because of the form in which relative track addresses are recorded, direct data sets to be accessed 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 this problem (Example 19), a tape containing 204 character records arranged in key sequence is used to create a direct data set. A 4-byte binary key for each record ranges from 1000 to 8999, so space for 8000 records is requested.

```
//DAOUTPUT DD     DSNAME=SLATE.INDEX.WORDS,DCB=(DSORG=DA,                C
//             BLKSIZE=200,KEYLEN=4,RECFM=F),SPACE=(204,8000),---
//TAPINPUT DD     ---
-------------------------------------------------------------------------
DIRECT     START
           ...
           L       9,=F'1000'
           OPEN    (DALOAD,(OUTPUT),TAPEDCB)
           LA      10,COMPARE
NEXTREC    GET     TAPEDCB
           LR      2,1
COMPARE    C       9,0(2)                  Compare key of input against  C
             control number
           BNE     DUMMY
           WRITE   DECB1,SF,DALOAD,(2)     Write data record
           CHECK   DECB1
           AH      9,=H'1'
           B       NEXTREC
DUMMY      C       9,=F'8999'              Have 8000 records been written?
           BH      ENDJOB
           WRITE   DECB2,SD,DALOAD,DUMAREA  Write dummy
           CHECK   DECB2
           AH      9,=H'1'
           BR      10
INPUTEND   LA      10,DUMMY
           BR      10
ENDJOB     CLOSE   (TAPEDCB,,DALOAD)
           ...
DUMAREA    DS      CL5
DALOAD     DCB     DSORG=PS,MACRF=(WL),DDNAME=DAOUTPUT,                   C
             DEVD=DA,SYNAD=CHECKER,---
TAPEDCB    DCB     EODAD=INPUTEND,MACRF=(GL), ---
```

Example 19. Creating a Direct Data Set

ADDING/UPDATING RECORDS ON A DIRECT DATA SET

The facilities and the techniques for adding records to a direct data set depend to a great extent on the format of the records and the organization used.

Format F With Keys: The add function is essentially an update by record identification. The reference to the record can be made by either a relative block address or a relative track address.

If you attempt to add a record by relative block address, the system converts the address to a relative track. 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. However, if you request the extended search option, the new record will be written in place of the first dummy record found within the search limit you specify. If none is found, you are notified that the write operation could not take place. In the same way, a reference by relative track address causes the record to be written in place of the first dummy record on that track or the first within the search limit, if requested.

Format F Without Keys: Here too, the add function is really an update of dummy records already in the data set. The main difference is that dummy records cannot be written automatically when the data set is created. You will have to use your own method for flagging dummy records. The update form of the WRITE macro instruction (MACRF=W) must be used rather than the add form (MACRF=WA).

You will have to retrieve the record first (READ macro instruction), test for a dummy record, update, and write.

Format V or U With Keys: The technique used to add records in this case depends on the way the data set is organized -- indirect addressing or cross-reference table. If indirect addressing is used to create the data set, 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 U Without Keys: This format does not lend itself to making additions. You can refer to a record only by its relative track or actual device address.

Tape-to-Disk Add -- Direct Data Set: This problem (Example 20) involves adding records to the data set created in the last example. Notice that the write operation adds the key and the data record to the data set. If the existing record is not a dummy record, an indication is returned in the exception code of the DECB. For that reason, it is better to use the WAIT macro instruction instead of the CHECK macro instruction to test for errors or exceptional conditions.

```
//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
          ...
```

Example 20. Adding Records to a Direct Data Set

Tape-to-Disk Update -- Direct Data Set: This problem (Example 21) is similar to Example 20. However, since you are updating, there is no check for dummy records. The existing direct data set contains 25,000 records whose 5-byte keys range from 00001 to 25,000. Each data record is 100 bytes long. The first 30 characters are to be updated. The input tape records are 35 characters long -- 5-byte key and 30-byte data. Notice that only data is brought into main storage for updating.

```
//DIRECTDD DD     DSNAME=SLATE.INDEX.WORDS,---
//TAPINPUT DD     ---
```

```
DIRUPDAT   START
           ...
           OPEN   (DIRECT,(UPDAT),TAPEDCB)
NEXTREC    GET    TAPEDCB,KEY
           PACK   KEY,KEY
           CVB    3,KEYFIELD
           SH     3,=H'1'
           ST     3,REF
           READ   DECBRD,DI,DIRECT,'S','S',0,REF+1
           CHECK  DECBRD
           L      3,DECBRD+12
           MVC    0(30,3),DATA
           ST     3,DECBWR+12
           WRITE  DECBWR,DI,DIRECT,'S','S',0,REF+1
           CHECK  DECBWR
           B      NEXTREC
           ...
KEYFIELD   DS     0D
           DC     XL3'0'
KEY        DS     CL5
DATA       DS     CL30
REF        DS     F
DIRECT     DCB    DSORG=DA,DDNAME=DIRECTDD,MACRF=(RISC,WIC),          C
               OPTCD=R,BUFNO=1,BUFL=100
TAPEDCB    DCB    ---
           ...
```

Example 21. Updating a Direct Data Set

Consideration for User Labels: User labels must be created when the data set is created. They may be updated when processing a direct data set but not added or deleted. When creating a multi-volume direct data set using BSAM, you should turn off the header exit entry after OPEN and turn on the trailer label exit entry just prior to issuing the CLOSE. This eliminates the end-of-volume exits. The first volume, containing the user label track, must be mounted at CLOSE time. If you have requested exclusive control, OPEN/CLOSE will use ENQ/DEQ facilities to prevent simultaneous reference to user labels.

Part 3: Data Set Disposition and Space Allocation

Allocating Space on Direct Access Volumes

When direct access storage space is required for a data set, you have to specify the amount of space needed and the device type. The operating system selects the device and allocates the space accordingly. This facility provides for more flexible and efficient use of devices and available storage space. It also relieves you of the responsibility and details involved in efficient space control.

Before a direct access volume can be used for data storage, it must be initialized by the utility program, Direct Access Storage Device Initialization (DASDI). The DASDI functions include in part:

- Creating the standard 80-byte volume label and writing it on cylinder 0, track 0, of the volume.
- Initializing the volume table of contents (VTOC). The location of the VTOC depends upon the conventions used by your installation when initializing the volume.
- Writing the home address (HA) and capacity record (R0) for each track.
- Checking tracks and making alternate track assignments if necessary.

When the data set is to be stored on a direct access volume, you must supply control information designating the amount of space to be allocated and in what manner. This information is supplied in the data definition (DD) statement for the data set.

SPECIFYING SPACE REQUIREMENTS

The amount of space required can be specified in terms of blocks, tracks, or cylinders. If you want to maintain device-independence across direct access device types, specify your space requirements in terms of blocks. Otherwise, if your request is in terms of tracks or cylinders, you must be aware of such device considerations as cylinder or track capacity.

Cylinder allocation allows faster input/output of sequential data sets than does track allocation. Track allocation stops input/output at the end of every track to prevent references on the same cylinder outside of the data set. The time difference occurs when you use the sequential access method or the partitioned access method to read a data set whose record format is not fixed standard (FS). If the data set is a partitioned data set, the time difference occurs both when loading a module from the data set and when reading the data set's directory.

Allocation by Blocks: When the amount of space required is expressed in terms of blocks, you must specify the number and average block length of the blocks within the data set, as in this example:

```
//  DD   --,SPACE=(300,(5000,100))

      300 = average block length in bytes.
     5000 = quantity (number of blocks).
      100 = increment (to be used if the quantity is not sufficient)
            allocated in terms of additional blocks.
```

Note: When average block and secondary space allocation are being used, the BLKSIZE parameter specified must be equal to the maximum block length.

From this information, the operating system estimates and allocates the number of tracks required. Space is always allocated in whole track units. 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 increment are large enough to contain the largest block to be written. Otherwise, all of the space requested is allocated but erased as the system tries to find a space large enough for the record.

Allocation by Tracks or Cylinders: When the amount of space required is expressed in terms of tracks or cylinders, you must also specify the device type in the DD statement, as in these examples:

```
//  DD   --,SPACE=(TRK,(100,5)),UNIT=2301
//  DD   --,SPACE=(CYL,(3,1)),UNIT=2311
```

Allocation by Absolute Address: If the data set contains location-dependent information in the form of an absolute track address, (MBBCCHHR), space should be requested in terms of the number of tracks and the beginning address, as in this example:

```
//  DD   --,SPACE=(ABSTR,(500,20)),UNIT=2311
```

where: 500 tracks are required beginning at relative track 20.

Additional Space Allocation Options: The DD statement provides you with a great deal of flexibility in specifying space requirements. You can request that the space be contiguous (CONTIG) or separated (SPLIT). These and other options are described in detail in the Job Control Language Reference manual.

ESTIMATING SPACE REQUIREMENTS

In order to determine how much space your data set requires, you must consider a number of variables:

- Device type.
- Track capacity.
- Tracks per cylinder.
- Cylinders per volume.
- Data length (block size).
- Key length.
- Device overhead.

Figure 36 lists the physical characteristics of a number of direct access storage devices.

Device Type	Volume Type	Track Capacity*	Tracks/Cylinder	No. of Cylinders	Total Capacity*
2311	Disk	3625	10	200	7,250,000
2314	Disk	7294	20	200	29,176,000
2302	Disk	4984	46	246	56,398,944
2303	Drum	4892	10	80	3,913,600
2301	Drum	20483	8	25**	4,096,600
2321	Cell	2000	20***	980***	39,200,000

*Capacity indicated in bytes.
**There are 25 logical cylinders in a 2301 Drum.
***A volume is equal to one bin in a 2321 Data Cell.

Figure 36. 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, R0, etc. Device overhead varies with each device and depends also on whether the blocks are written with keys. To compute the actual space required for each block including device overhead, you can use the formulas in Figure 37. Note that any fraction of a byte must be treated as an extra byte. For example, if the formulas give 15.067 bytes, you must allocate 16 bytes.

Device	Bytes Required by Each Data Block			
	Blocks With Keys		Blocks Without Keys	
	Bi	Bn	Bi	Bn
2311	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2314	146+1.043(KL+DL)	45+KL+DL	101+1.043(DL)	DL
2302	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2303	146+KL+DL	38+KL+DL	108+DL	DL
2301	186+KL+DL	53+KL+DL	133+DL	DL
2321	100+1.049(KL+DL)	16+KL+DL	84+1.049(DL)	DL

Bi is any block but the last on the track.
Bn is the last block on the track.
DL is data length.
KL is key length.

Figure 37. Direct Access Device Overhead Formulas

The formulas can be combined in the following way:

If you intend to specify your space requirements in terms of 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 volume table of contents will reduce the space available on the rest of the volume.

On the other hand, if you specify your space requirements in terms of average block length, the system performs the computations for you.

Because a sequential data set and a direct data set are created in the same way, the estimate and specification of space requirements are identical. If you use the WRITE SZ macro instruction, your secondary allocation for a direct data set should be at least two tracks. Space allocation for a partitioned data set requires that you also consider the space used for the directory. Similarly, allocation for an indexed sequential data set requires that you consider the space needed for the prime area, index areas, and overflow areas.

ALLOCATING SPACE FOR A PARTITIONED DATA SET

What is the average size of the members to be stored on your direct access volume? How many members will fit on the volume? Will you need directory entries for the member names only or will aliases be used? How many? Will members be added or replaced frequently? All of these questions must be answered if you are to estimate your space requirements accurately and use the space efficiently. Note, too, that a partitioned data set cannot extend beyond one volume.

If your data set will be quite large, or you expect to do a lot of updating, it might be best to allocate a full volume. If it will be small or seldom subject to change, you should make your estimate as accurate as possible to avoid wasted space or wasted time used for recreating the data set.

Because the characteristics of all the members of the data set must be uniform, the record format could be specified as undefined (RECFM=U) and the block size (BLKSIZE) as a maximum length. It is a good practice to indicate a block length equal to track capacity, for example, BLKSIZE=3625 for a 2311 disk. You might then ask for either 200 tracks, or 20 cylinders, thus allowing for 725,000 bytes of data.

Assuming an average length of 70,000 bytes for each member, you need space for at least 10 directory entries. If each member also has an average of three aliases, space for an additional 30 directory entries is required.

Space for the directory is expressed in terms of 256-byte blocks. Each block contains from three to twenty entries, depending on the length of the user data field. If you expect 40 directory entries, request at least eight blocks. Because the space for the directory is allocated in full track units, any unused space on the track is wasted unless there is enough space left to contain a block of the first member. Therefore, the most advisable request in this case would be for 10 blocks.

Putting the space estimates into specification form, any of the following would cause the same allocation:

```
SPACE=(3625,(200,,10))
SPACE=(CYL,(20,,10))
SPACE=(TRK,(200,,10))
```

Although an increment has been omitted in these examples, it could have been supplied to provide for extension of the member area. The directory size, however, cannot be extended.

ALLOCATING SPACE FOR AN INDEXED SEQUENTIAL DATA SET

An indexed sequential data set can be divided into three areas: prime, index, and overflow. Space for these areas can be subdivided and allocated in several different ways:

- Prime area -- If you request space in terms of 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 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 need not be of the same device type as the prime area volume. If a separate index area is requested, you cannot catalog the data set with a DD statement.

  A slight variation for requesting an index area can be used if the total space occupied by the prime area and index area does not exceed one volume. In this case, you can request that the separate index area be embedded in the middle of 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 in terms of prime and index areas only, the system will automatically use any space remaining on the last cylinder used for master and cylinder indexes for overflow, provided the index area is on the same type of device as the prime area.

- Overflow area -- Although you can request an independent overflow area, it must be contained within one volume. If no specific request for index area is made, then it will be allocated from the specified independent overflow area.

  To request that a designated number of tracks on each cylinder be used for cylinder overflow records, you must use the CYLOFL parameter of the DCB macro instruction. The number of tracks that you can use on each cylinder equals the total number of tracks on the cylinder minus the sum of the tracks needed for track index and the tracks required for prime data, that is:

  Usable tracks =

  total tracks - (track index tracks + prime data tracks)

Note that when you create a one-cylinder data set, ISAM reserves one track on the last cylinder for the end-of-file filemark.

When requesting space for an indexed sequential data set, the DD statement must follow a number of conventions, as shown below and summarized in Figure 38.

Criteria			Restrictions on Unit Types and Number of Units Requested	Resulting Arrangement of Areas
1. Number of DD Statements	2. Types of DD Statements	3. Index Size Coded?		
3	INDEX PRIME OVFLOW	-	None	Separate index, prime, and overflow areas.
2	INDEX PRIME	-	None	Separate index and prime areas.
2	PRIME OVFLOW	No	None	Prime area and overflow area with an index at its end.
2	PRIME OVFLOW	Yes	The statement defining the prime area cannot request more than one unit.	Prime area and embedded index, and overflow area.
1	PRIME	No	None	Prime area with index at its end. Any unused index area will be used for independent overflow.
1	PRIME	Yes	Statement cannot request more than one unit.	Prime area with embedded index area.

Figure 38. Requests for Indexed Sequential Data Sets

- Space (SPACE) can be requested only in terms of cylinders (CYL) or absolute tracks (ABSTR). If the absolute track technique is used, the designated tracks must encompass an integral number of cylinders.

- Data set organization (DSORG) must be specified as indexed sequential (IS or ISU) in both the DCB macro instruction and the DCB parameter of the DD statement.

- All required volumes must be mounted when the data set is opened, that is, volume mounting cannot be deferred.

- If your prime area extends beyond one volume, you must indicate the number of units and volumes to be spanned, for example, UNIT=(2311,3),VOLUME=(,,,3).

- You can catalog the data set using the DD statement parameter DISP=(,CATLG) only if the entire data set is defined by one DD statement, that is, 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 from the prime area.

Note: If an error is encountered when allocating a multivolume data set, the IEHPROGM utility program should be used to scratch the data set control blocks 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 the Utilities manual.

SPECIFYING A PRIME DATA AREA

To request that the system allocate space and subdivide it as required, you should code:

```
//ddname   DD  DSNAME=dsname,DCB=DSORG=IS,                          C
//             SPACE=(CYL,quantity,,CONTIG),UNIT=unitname,          C
//             DISP=(,KEEP),---
```

You can accomplish the same type of allocation by qualifying your dsname with the element indication (PRIME). This element is assumed if omitted. It is required only if you request an independent index or overflow area. To request an embedded index area when an independent overflow area is specified, you must indicate DSNAME=dsname(PRIME). To indicate the size of the embedded index, you specify SPACE=(CYL, (quantity,,index size)).

SPECIFYING A SEPARATE INDEX AREA

In order 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),---
//            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),---
//            DSNAME=dsname(PRIME),---
//            DSNAME=dsname(OVFLOW),---
```

CALCULATING SPACE REQUIREMENTS FOR AN INDEXED SEQUENTIAL DATA SET

To determine the number of cylinders required for an indexed sequential data set, you must consider the number of blocks that will fit on a cylinder, the number of blocks that will be processed, and the

amount of space required for indexes and overflow areas. In making the computations, consider additional space that is required for device overhead as shown in the Figure 37. Remember the formula:

$$\text{Blocks per track} = 1 + \frac{\text{Track capacity - Length of the last block}}{\text{Length of other blocks}}$$

$$Bt = 1 + ((Ct-Bn)/Bi)$$

Step 1

Once you know how many records will fit on a track and the maximum number of records you expect to create, you can determine how many tracks you will need for your data.

$$\text{Number of tracks required} = \frac{\text{Maximum number of blocks}}{\text{Blocks per track}} + 1$$

ISAM load mode reserves the last prime data track for the filemark.

Example: Assume the existence of a 200,000 record part-of-speech dictionary to be stored on an IBM 2311 Disk Storage Unit as an indexed sequential data set. Each record in the dictionary has a 12-byte key (the word itself) and an 8-byte data area containing a part-of-speech code and control information. Each block contains 50 records -- LRECL=20 and BLKSIZE=1000. Using the formula from Table 16, we find that each track will contain 3 blocks or 150 records. A total of 1333 1/3 tracks will be required for the dictionary.

$$Bt = 1 + \frac{3625 - (20 + 12 + 1000)}{81 + 1.049(12 + 1000)} = 1 + \frac{2593}{1143} = 3$$

$$\text{Records per track} = (3 \text{ blocks})(50 \text{ records per block}) = 150$$

$$\text{Prime data tracks required (T)} = \frac{200{,}000 \text{ records}}{150 \text{ records per track}} + 1 = 1334\ 1/3$$

Step 2

You will want to anticipate the number of tracks required for cylinder overflow areas. The computations formula is the same as for prime data tracks, but you must remember that overflow records are unblocked and a 10-byte link field is added. Remember, if you exceed the space allocated for any cylinder overflow area, an independent overflow area is required. Those records are not placed in another cylinder overflow area.

$$\text{Overflow records per track} = 1 + \frac{\text{Track capacity - Length of last overflow record}}{\text{Length of other overflow records}}$$

$$Ot = 1 + ((Ct-Rn)/Ri)$$

Example: Approximately 5000 overflow records are expected for the data set described in step 1. Since 29 overflow records will fit on a track, 173 overflow tracks are required. This is approximately 2 overflow tracks for every 15 prime data tracks. Since the 2311 disk has 10 tracks per cylinder, it would probably be best to allocate 2 tracks per cylinder for overflow.

$$Ot = 1 + \frac{3625 - (20 + 12 + 20 + 10)}{81 + 1.049(12 + 20 + 10)} = 1 + \frac{3563}{126} = 29$$

$$\text{Overflow tracks required} = \frac{\text{5000 records}}{\text{29 records per track}} = 173$$

Overflow tracks per cylinder (Oc) = 2

Step 3

You will have to set aside space in the prime area for track index entries. There will be two entries (normal and overflow) for each track on a cylinder that contains prime data records. The data field of each index entry is always 10 bytes. The key length corresponds to the key length for the prime data records. How many index entries will fit on a track?

$$\text{Index entries per track} = 1 + \frac{\text{Track capacity - Length of last index entry}}{\text{Length of other index entries}}$$

$$It = 1 + ((Ct-En)/Ei)$$

Example: Again assuming a 2311 disk and records with a 12-byte key, we find that 35 index entries will fit on a track.

$$It = 1 + \frac{3625 - (20 + 12 + 10)}{81 + 1.049(12 + 10)} = 1 + \frac{3583}{105} = 1 + 34 = 35$$

Step 4

The number of tracks required for track index entries will depend on the number of tracks per cylinder and the number of track index entries per track. Any unused space on the last track of the track index can be shared with prime data records if they will fit.

$$\text{Number of track index tracks per cylinder} = \frac{2(\text{Tracks per cyl.} - \text{overflow tracks per cyl.}) + 1}{\text{Index enteries per track} + 2}$$

$$Ic = 2(Tc-Oc)+1/(It+2)$$

Note: For variable-length records or when a prime data record will not fit on the last track of the track index, the last track of the track index is not shared with prime data records. In such a case, if the remainder of the division is less than or equal to 2, do not round the quotient up to the nearest integer. In all other cases, round the quotient up to the nearest integer.

Example: The 2311 disk has 10 tracks per cylinder. You can fit 35 track index entries per track. Therefore, you need less than one track for each cylinder:

$$Ic = \frac{2(10-2) + 1}{35 + 2} = \frac{17}{37}$$

The space remaining on the track is ((1-17/37)(3625)) = 1960 bytes. This is enough for one block of prime data records. Since the normal number of blocks per track is 3, the block uses one third of the track, and the effective value of Ic is therefore 1-1/3 = 2/3.

Step 5

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

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

$$Pc = Tc - Oc - Ic$$

Example: If you set aside 2 cylinder overflow tracks, and you require 2/3 of a track for the track index, 7 1/3 tracks are available on each cylinder for prime data records.

$$Pc = 10 - 2 - 2/3 = 7\ 1/3$$

Step 6

The number of cylinders required for the prime data records, track index area, and cylinder overflow area is determined by the number of prime data tracks required divided by the number of prime data tracks available on each cylinder.

$$\text{Number of cylinders required} = \frac{\text{Prime data tracks required}}{\text{Prime data tracks per cylinder}}$$

$$C = T/Pc$$

Example: You need 1333 1/3 tracks for prime data records. You can use 7 1/3 tracks per cylinder. Therefore, 182 cylinders are required for your prime area and cylinder overflow areas.

$$C = \frac{1333\ 1/3}{7\ 1/3} = 181.9$$

Step 7

You will need space for a cylinder index as well as track indexes. There is a cylinder index entry for each track index, i.e., 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.

$$\text{Number of tracks required for cylinder index} = \frac{\text{Track indexes} + 1}{\text{Index entries per track}}$$

$$Ci = (C+1)/It$$

Example: You have 182 track indexes. Since 35 index entries fit on a track, you need 5.3 tracks for your cylinder index. The remaining space on the last track is unused.

$$Ci = \frac{182 + 1}{35} = 5.3$$

Note that every time a cylinder index crosses a cylinder boundary, ISAM writes a dummy index entry that lets ISAM chain the index levels together. The addition of dummy entries can increase the number of tracks required for a given index level. To determine how many dummy entries will be required, divide the total number of tracks required by the number of tracks on a cylinder. If the remainder is 0, subtract 1 from the quotient. If the corrected quotient is nonzero, 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 in the middle of a cylinder.

Step 8

If you have a data set large enough to require master indexes, you will want to calculate the space required according to the number of tracks for master indexes (NTM parameter) you specified in the DCB macro instruction or the DD statement.

If the cylinder index exceeds the NTM specification, an entry is made in the master index for each track of the cylinder index. If the master index itself exceeds the NTM specification, a second level master index is started. Up to three levels of master indexes are created if required.

The space requirements for the master index are computed in the same way as the cylinder index.

Number of tracks required for master indexes = $\frac{\text{Cylinder index tracks}}{\text{Index entries per track}} + 1$

$M_1 = (Ci+1)/It$ when $Ci>NTM$

$M_2 = (M_1+1)/It$ when $M_1>NTM$

$M_3 = (M_2+1)/It$ when $M_2>NTM$

Example: Assume that your cylinder index will require 22 tracks. Since large keys are used, only 10 entries will fit on a track. Assuming that NTM was specified as 2, 3 tracks will be required for a master index, and two levels of master index will be created.

$M_1 = (22+1)/10 = 2.3$

Note that every time a master index crosses a cylinder boundary, ISAM writes a dummy index entry that lets ISAM chain the index levels together. The addition of dummy entries can increase the number of tracks required for a given index level. To determine how many dummy entries will be required, divide the total number of tracks required by the number of tracks on a cylinder. If the remainder is 0, subtract 1 from the quotient. If the corrected quotient is nonzero, 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 in the middle of a cylinder.

Summary: Indexed Sequential Space Requirement Calculations

1. How many blocks will fit on a track?

   Bt = 1 + ((Ct-Bn)/Bi)

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

   Ot = 1 + ((Ct-Rn)/Ri)

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

   It = 1 + ((Ct-En)/Ei)

4. How many track index tracks are needed per cylinder?

   Ic = 2(Tc-Oc)+1/(It+2)

5. How many tracks on each cylinder can be used for prime data records?

   Pc = Tc - Oc - Ic

6. How many cylinders are needed for the prime data area?

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

7. How many tracks are required for the cylinder index?

   Ci = (C+1)/It

8. How many tracks are required for master indexes?

   M = (Ci+1)/It

Control and Disposition of Data Sets

There are two levels of status and disposition of the data sets you use for your processing. The status and disposition information must be provided to the system in the disposition field of the DD statement, DISP=(status,disposition). The first level deals with the status of the data set when you begin processing and the relationship of the data set to other job steps in your job or other jobs. The second deals with what is to be done with the data set when you have completed processing. It is at this level of control and disposition that you can take advantage of the cataloging facilities of the operating system.

A data set that is being used for input has a status of OLD. If it can be used by more than one job, the status should be specified as SHR. If you are going to add to the input data set, specify MOD. The system automatically positions the access mechanism after the last record when the data set is opened. A NEW output data set should be so indicated.

Having identified the status of the data set at the beginning of your job step, you should specify how you want it disposed of at the end of processing. If the disposition is to be unchanged, you need not specifyanything more. The status of an existing data set remains unchanged; a new data set is deleted.

The requested disposition is performed at the end of the job step. A data set to be used in a later job can be kept (KEEP) until a subsequent request is made to DELETE it. If the data set is to be used by more than one job step in the same job, you can specify that it is to be passed (PASS).

The most useful disposition provided by the system is the cataloging facility (CATLG). The data set name is recorded by the system and its volume noted. An old data set can subsequently be removed from the catalog if you so request (UNCATLG).

If you wish, you can specify one disposition to be performed if the job step terminates normally, and a different disposition to be performed if the job step terminates abnormally. For example, you can specify DISP=(OLD,DELETE,KEEP) if you wish to delete a data set under normal conditions, but wish to keep it if processing is abnormally terminated. For normal termination, you can specify any disposition -- PASS, KEEP, DELETE, CATLG, or UNCATLG; for abnormal termination, you can specify any disposition except PASS.

ROUTING DATA SETS THROUGH THE OUTPUT STREAM

Data sets that are to be printed or punched can be routed through the output stream. This allows greater flexibility in scheduling print and punch operations, and improves operating system efficiency.

When you route a data set through the output stream, you do not request a unit record device for exclusive use by your job step. Instead, you assign the data set to an output class, which may include data sets from many different jobs. Output classes are identified by the letters A-Z and the digits 0-9. Each is associated with a specific device type. By convention, class A consists of high priority output to be printed (for example, a listing of job control statements), and class B consists of output to be punched. Other classes are defined by the installation.

To route a data set through the output stream, and to assign it to class A, you would simply code SYSOUT=A in the DD statement. No other parameters are necessary. A description of other parameters that can be coded appears in the Job Control Language manuals.

In a system with the primary control program, you write a SYSOUT data set directly onto the system output device. This device is assigned to the output class by the operator; it may be either a unit record device or a magnetic tape unit. If it is a tape unit, the operator is responsible for transcribing the tape on a punch or printer. Depending on the output class and on the installation, the tape may be transcribed during a later shift or on a smaller, offline computing system.

In a system with MFT or MVT, you write a SYSOUT data set in one of two ways, as determined by the operator. Either you write the data set directly onto the system output device, or you write the data set into intermediate storage on a direct access device. In the latter case, a system output writer automatically copies your data set onto the system output device after your job has been completed. The system output device can be either a unit record device or a magnetic tape unit, as is true in systems with PCP.

Note: The following discussion assumes that, for systems with MFT and MVT, system output data sets are written into intermediate storage and copied by a SYSOUT writer. When these data sets are written directly onto the system output device, they are handled as described for systems with PCP.

OPENING A SYSOUT DATA SET

You open and close a SYSOUT data set in the same way as any other data set. If specified in an exit list, the data control block exit routine is entered in the usual manner. An exit list should not specify user label exits, because you cannot write labels on a unit record device.

If you observe certain restrictions, which are indicated below, you can create several SYSOUT data sets during a single job step.

Data Sets That Are Open Concurrently: In a system with the primary control program, only one of a group of concurrently open data sets can be assigned to an output class for which the SYSOUT device is a magnetic tape unit. If necessary, you can assign data sets to class A and a maximum of seven other output classes.

If a punch or printer is used as a SYSOUT device, any number of data sets can be assigned to one output class. The data control blocks for all these data sets can refer to the same DD statement. When printed or

punched, the data sets appear as a single data set because their records form a single chronological sequence.

In a system with MFT or MVT, SYSOUT data sets must always be defined by separate DD statements. They can be assigned to the same output class or to different output classes. There is no special restriction on the number of output classes that can be used.

Data Sets That Are Not Open Concurrently: To avoid having two data sets open concurrently, you can open and close each data set as often as required. In a system with the primary control program, records of the two data sets will form a single chronological sequence if the data sets belong to the same output class.

In a system with MFT or MVT, records of two data sets do not form a chronological sequence, because you write each data set into a separate area of intermediate direct access storage. When copied by a SYSOUT writer, data sets are written on the SYSOUT device in the order of their DD statements.

WRITING A SYSOUT DATA SET

To create a SYSOUT data set, you can use either the basic sequential or the queued sequential access method. You can write records in any format defined for the type of unit record device on which the data set is to be written. Record length must not exceed the maximum allowable for the device.

Under MFT or MVT, when you use the queued sequential access method (QSAM) with fixed blocked records or the basic sequential access method (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 the user's LRECL is not known to the system until execution time. 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. For further information, refer to "Creating Data Sets in the Output Stream" in the Job Control Language User's Guide.

Because a SYSOUT data set may be written on a magnetic tape or direct access device, it must be device-independent. You should therefore omit the device dependency operand in the DCB macro instruction, or should code it as DEVD=DA.

Your SYNAD routine is entered on errors that occur when the data set is first written. In a system with the primary control program, it is entered when you write the data set on the system output device. In a system with MFT or MVT, it is entered when you write the data set into intermediate storage on a direct access device.

Your program is responsible for printing format, pagination, and header control. Use of control characters must be indicated in the usual way in the data control block. If you do not use control characters, a standard control is supplied in systems with MFT or MVT. When channel 12 is sensed, a printer will space one line and skip to channel 1; a card punch will select punch pocket 1.

In a system with MFT or MVT, cards can be punched only in EBCDIC mode.

CONCATENATING SEQUENTIAL AND PARTITIONED DATA SETS

Two or more sequential or partititioned data sets can be automatically retrieved by the system and processed successively as a single data set. This reading technique is known as concatenation. A maximum of 255 data sets (16, if partitioned) can be concatenated, but they must be used only for input. To save time when processing two consecutive data sets on a single volume, you specify LEAVE in your OPEN macro instruction. Concatenated data sets cannot be read backwards.

When data sets are concatenated, the system treats the group as a single data set and only one data extent block (DEB) is constructed. Thus, it is important to consider the characteristics of the individual data sets which are being concatenated. Data sets with like characteristics are those which may be processed correctly using the same data control block (DCB), input/output block (IOB), and channel program. Any exception makes them "unlike". The system must be informed if "unlike" data sets are concatenated. This is accomplished by modifying the DCBOFLGS field of the data control block. The indication must be made before the end of the current data set is reached. You must set bit 4 to 1 by using the instruction OI DCBOFLGS,X'08' as described in "Modifying the Data Control Block." If bit 4 of the DCBOFLGS field is 1, end-of-volume processing for each data set will issue a CLOSE for the data set just read and an OPEN for the next concatenated data set. This opening and closing procedure updates the fields in the DCB and, if necessary, builds a new IOB and a new channel program. If the buffer pool was obtained automatically by the open routine, the procedure also frees the buffer pool and obtains a new one for the next concatenated data set. The procedure does not issue a FREEPOOL for the last concatenated data set. Unless you have some way of determining the characteristics of the next data set before it is opened, you should not reset the DCBOFLGS field to indicate "like" characteristics during processing.

When "unlike" data sets have been concatenated, we urge that you do not issue multiple input requests, that is, a series of READ or GET macro instructions in your program. Otherwise, you will have to arrange some way to determine which requests have been completed and which must be reissued. In any case, the GET or READ macro instruction that detected the end of data set will have to be reissued. Figure 39 illustrates a possible routine for determining when a GET or READ must be reissued. This restriction does not apply to "like" data sets since no open or close operation is necessary between data sets.

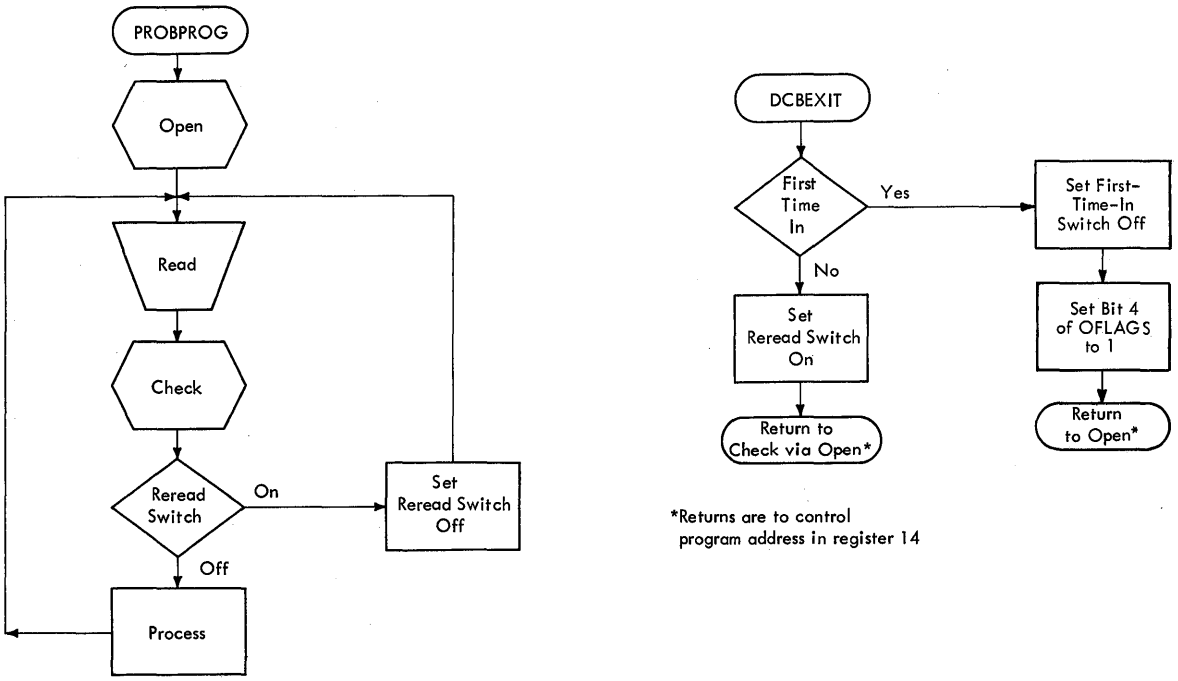


Figure 39. Reissuing a READ for "Unlike" Concatenated Data Sets

When the change is made from one data set to another, 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. An exception to this arises with partitioned data sets. Your EODAD routine receives control at the end of each member. At that time, you can process the next member or close the data set.

Further discussion and examples of concatenated data sets are contained in the Job Control Language Reference manual.

CATALOGING DATA SETS

To provide the cataloging facilities of the operating system, a catalog is created that is itself a data set residing on one or more direct access volumes. It is organized into levels of indexes that connect the data set names to corresponding volumes and data set sequence numbers. For each level of qualification in the data set name, there is an index group in the catalog.

The highest level of the catalog resides on the system residence volume. The volume table of contents (VTOC) contains an entry for the data set control block (DSCB) defining the catalog and its highest level index, the volume index. The lowest level index contains the simple name of the data set and the number of the volume on which it resides.

The complete catalog can exist on the system residence volume, or you can specify that parts of it be constructed on other volumes. Any volume containing part of the catalog is called a control volume. The use of control volumes allows data sets that are functionally related to be cataloged separately. There are several advantages:

- Control volumes can be moved from one processing system to another.

- System residence requirements can be reduced by placing seldom used indexes on a control volume.

For any given data set, only one level of control volume, other than the system residence volume, can be used. Notice that in Figure 34, INDEX E, which is the highest level on the control volume, has an entry in both volume indexes.

The same type of cataloging facilities are available for maintaining generation data groups. Cataloging each new generation with a unique name would be both inconvenient and inefficient. By cataloging individual data sets in a chronological collection by number, the entire collection can be stored under a single data set name.

Each update of the data set is called a generation; the number associated with it is called a generation number. A generation data group is the entire collection of chronologically related data sets that can be referred to by the same data set name. A particular generation can be referred to by either the absolute generation name or relative generation number of the data set.

ABSOLUTE GENERATION NAME: The operating system assigns each data set in the generation data group an absolute generation name in the form GggggVvv:

- gggg is an unsigned, four digit, decimal generation number.
- vv is an unsigned, two digit, decimal version number.

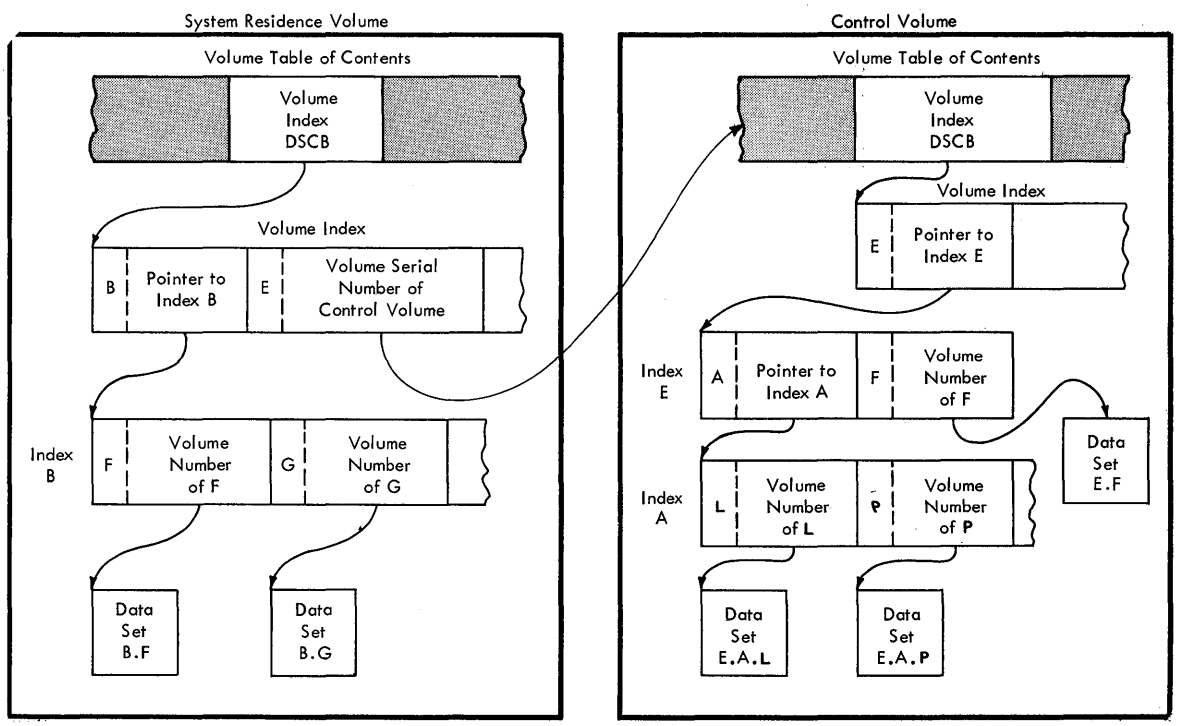


Figure 40. Catalog Structure on Two Volumes

The generation number indicates how far removed the data set is from the original generation. The version number indicates how many times the associated generation has been replaced. Only the most recent version of a specific generation is retained.

Generation Increment: You can specify the increment by which the generation number is changed. For example, if you request a current generation G0013V04 and an increment of 2, the new generation would be assigned the absolute generation name G0015V00.

Version Increment: When you replace the same generation with a new version, it is your responsibility to assign the new, nonzero version number.

CONCATENATED GENERATIONS: You can request a concatenation of all existing data sets in the generation data group, starting with the most recent and ending with the oldest, by specifying only the data set name.

RELATIVE GENERATION NUMBER: Rather than request a data set by its absolute generation number, you can refer to it relative to the most recent generation, that is, DSNAME=dsname(0). Those immediately preceding the most recent are then identified as -1, -2, etc. New generations are created by referring to them as DSNAME=name(+1),(+2),(+3), etc. The last of these is cataloged as (0) and the other generations in the catalog are adjusted accordingly at the end of the job.

ENTERING A DATA SET NAME IN THE CATALOG

The catalog structure, including all levels of indexes, is initially created or modified by the system utility program IEHPROGM. A data set name can then be entered if the proper index levels of the name exist.

For example, if a data set named A.B.C is to be cataloged, the volume index on the system residence volume must have an index entry for index A, which must point to an index B. When the data set A.B.C is cataloged, C is entered into index B along with the volume serial number where data set A.B.C resides. The cataloging request is entered as:

```
//ddname DD DSNAME=A.B.C,DISP=(,CATLG)
```

ENTERING A GENERATION DATA GROUP IN THE CATALOG

A data set that is part of a generation data group is represented in the catalog by an additional level of indexing that contains an entry for each generation. The system utility program IEHPROGM is used to create the index levels and to instruct the system as to how the generations are to be maintained.

CONTROL OF CONFIDENTIAL DATA -- PASSWORD PROTECTION

In addition to the usual label protection that prevents opening a data set without the correct data set name, the operating system provides a data set security facility that prevents 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 TSO jobs, by the TSO terminal 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, the job is terminated by the system.

You can request password protection when the data set is created. The system sets the data set security byte in the Standard Data Set Label 1 as shown in the Tape Labels publication. Once security protection has been requested, it cannot be removed without recreating the data set and scratching the protected data set. Security protection can be added to an old data set or changed from NOPWREAD to PASSWORD if the data set is opened for OUTPUT or OUTIN when it is opened for the first time during a job step.

Each protected data set has at least one entry in a catalog named PASSWORD that must be created on the system residence volume. Each entry in the password data set consists of a 44-byte data set name field and an 8-byte password field. The next 80-byte record contains a 2-byte binary counter that is incremented by one each time the protected data set is opened successfully. The third byte is used to indicate that the processing program can read, write, or both read and write records on the protected data set. The remaining 77 bytes can be used at the discretion of your installation.

The password data set can also be protected by a master password contained in one of its entries. A complete description of password protection is contained in the System Programmer's Guide.

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 41). In the case of direct access volumes, the data set label group is the data set control block (DSCB).

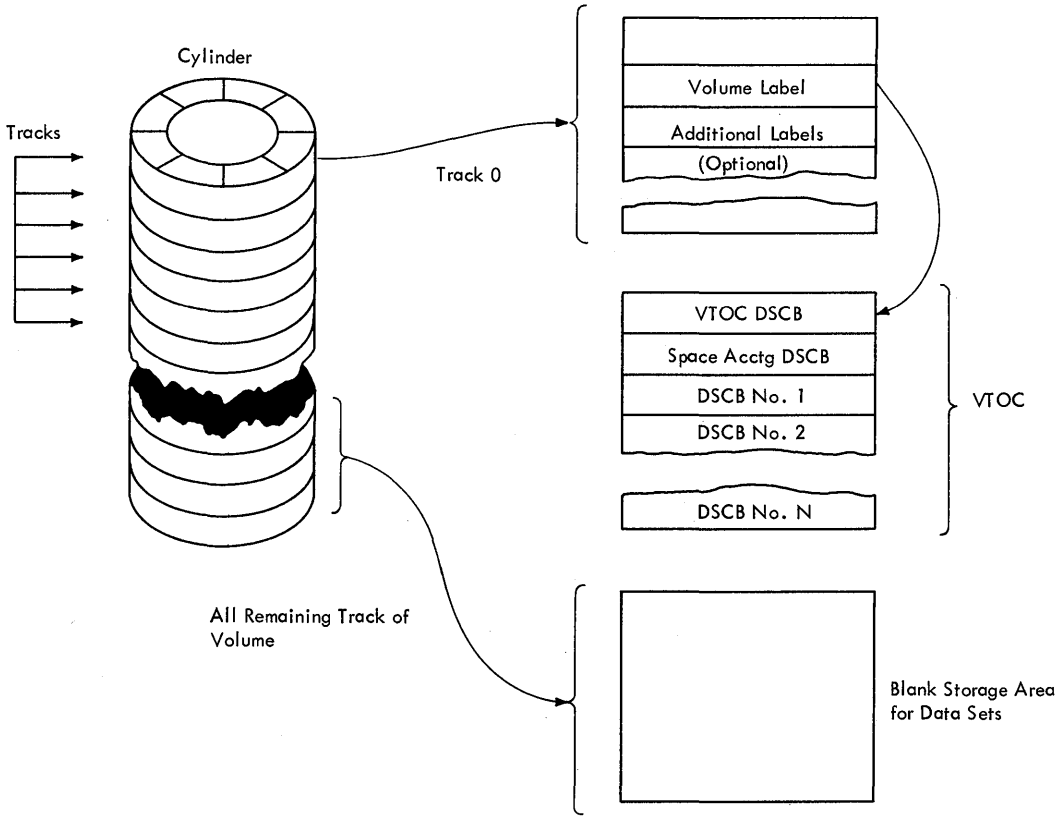


Figure 41. Direct Access Labeling

Volume Label Group

The volume label group immediately follows the initial program loading (IPL) records on track 0 (of cylinder 0) of the volume. It consists of the initial volume label plus a maximum of seven additional volume labels. The initial volume label identifies a volume and its owner, and is used to verify that the correct volume is mounted. It can also be used to prevent use of the volume by unauthorized programs. The additional labels are processed by means of an installation routine that is incorporated into the system.

The format of the direct access volume label group is shown in Figure 42.

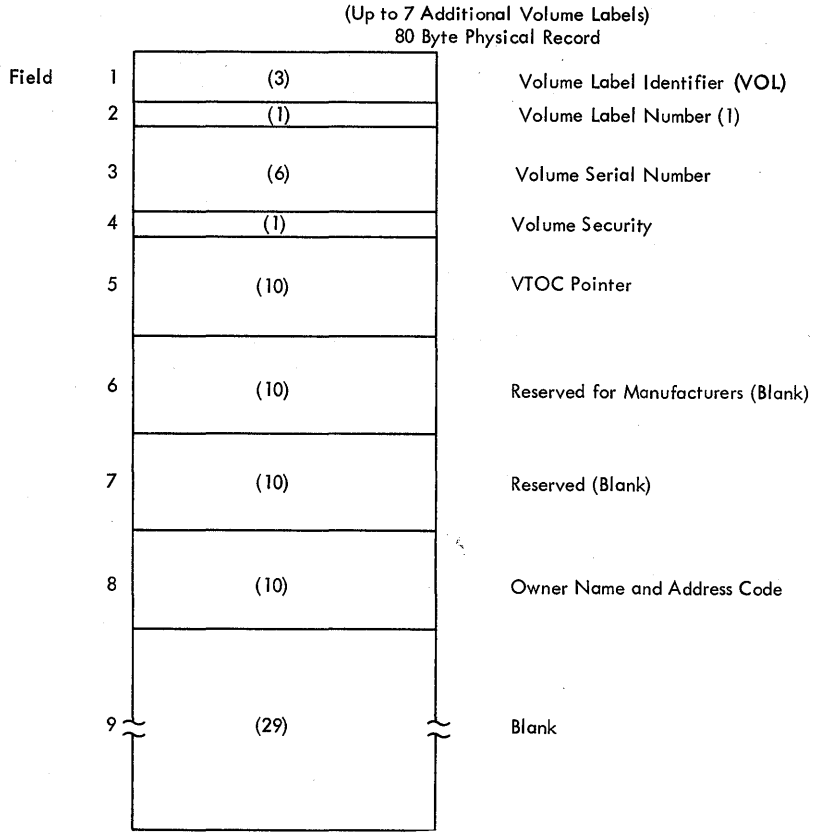


Figure 42. Initial Volume Label

Direct Access Volume Label Format

Volume Label Identifier (VOL): Field 1 contains the initial volume label.

Volume Label Number (1): Field 2 identifies the relative position of the volume label in a volume label group. It must be written as 1.

The operating system identifies an initial volume label when, in reading the initial record, it finds that the first four characters of the record are VOL1.

Volume Serial Number: Field 3 contains a unique identification code assigned when the volume enters the system. You can place the code on the external surface of the volume for visual identification. The code is normally numeric (000001-999999), but may be any six alphameric characters.

Volume Security: Field 4 is reserved for future use by installations that wish to provide security at the volume level. It must be written as 0.

VTOC Pointer: Field 5 of direct access volume label 1 contains the address of the volume table of contents (VTOC).

Reserved for Manufacturers: Field 6 is reserved for future standardization purposes. Leave it blank.

Reserved: Field 7 is reserved for future developmental purposes. Leave it blank.

Owner Name and Address Code: Field 8 contains a unique identification of the owner of the volume.

All of the bytes in Field 9 are left blank.

Data Set Control Block (DSCB) Group

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 a corresponding data set control block to describe its characteristics. The DSCB appears in the volume table of contents (VTOC) and contains operating system data, device-dependent information, and data set characteristics, in addition to space allocation and other control information. The format of the DSCB is illustrated in the System Control Blocks manual.

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 43.

Each group can include up to eight labels, but the space required for both groups must not be more than one 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 problem program recording the data set. The first four 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 four characters must be UTL1,...., UTL8.

User Header and Trailer Label Format

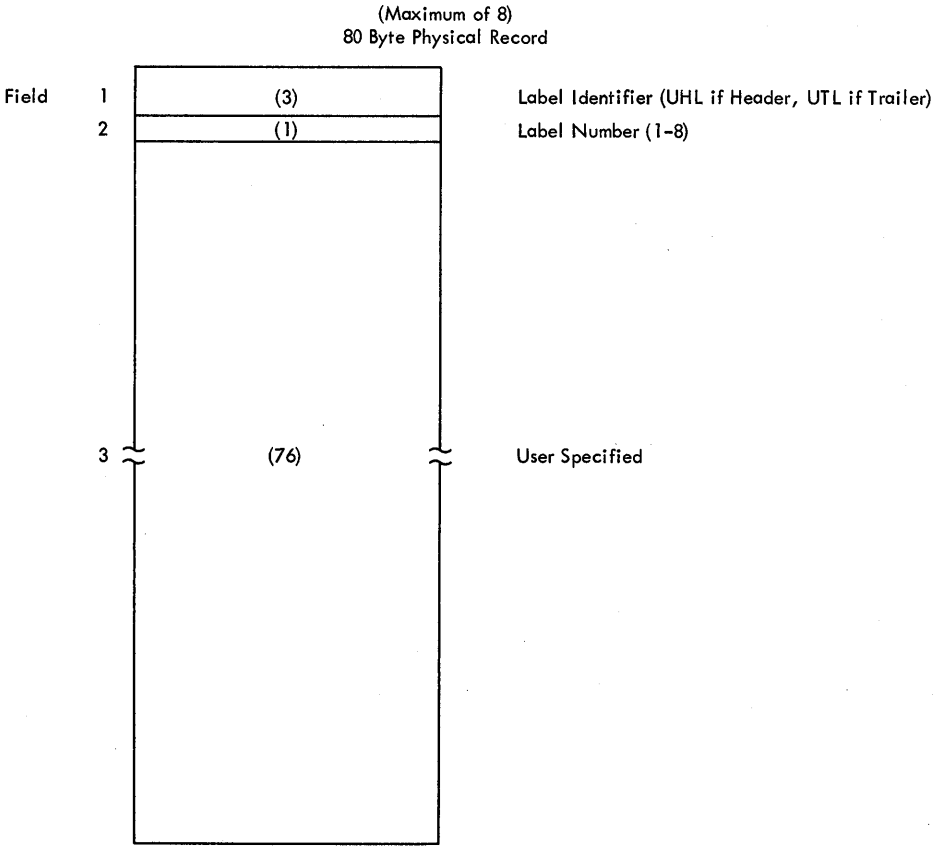


Figure 43. User Header and Trailer Labels

Label Identifier: Field 1 indicates the kind of user header label. UHL indicates a user header label; UTL indicates a user trailer label.

Label Number: Field 2 identifies the relative position (1-8) of the label within the user label group.

User Specified: Field 3 (76 bytes).

Appendix B: Control Characters

As an optional feature, all record formats may include a control character in each logical record. This control character will be recognized and processed if a data set is being written to a printer or punch.

For format-F and -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 data control block, the character must appear in every record and other line spacing or stacker selection options also specified in the data control block are ignored.

Machine Code

You can specify in the data control block that the machine code control character has been placed in each logical record. If the record is to be written, the appropriate byte must contain the command code bit configuration specifying both the write and the desired carriage or stacker select operation. If the record is not to be written, the byte can specify any command other than write.

Command codes for specific devices are contained in IBM System Reference Library publications describing the control units or devices.

Extended American National Standards Institute Code

In place of machine code, you can specify control characters defined by the American National Standards Institute, Inc. (ANSI). These characters must be represented in EBCDIC.

The extended American National Standards Institute Code (ANSI) is as follows:

Code	Action Before Writing Record
b	Space one line before printing (blank code)
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7
8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12
V	Select punch pocket 1
W	Select punch pocket 2

These control characters include those defined by ANSI FORTRAN. If any other character is specified, it is interpreted as 'b' or V, depending on the device being used; no error indication is returned.

Index

Indexes to Systems Reference Library publications are consolidated in IBM System/360 Operating System: Systems Reference Library Master Index, Order Number GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

C

D

E

F

G

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/360 Operating System
Data Management Services

Order Number GC26-3746-0

Your comments about this publication will help us to produce better publications for your use. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes ☐

No ☐

Name _____

Job Title _____

Address _____

_____ Zip _____

No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This publication is one of a series that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Each reader's comment will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. D78

fold fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/360 Operating System
Data Management Services

Order Number GC26-3746-0

Your comments about this publication will help us to produce better publications for your use. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes ☐

No ☐

Name _____

Job Title _____

Address _____

_____ Zip _____

No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This publication is one of a series that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Each reader's comment will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. D78

fold fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]