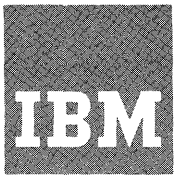
File No. S360-30
Form C28-6537-1

OS



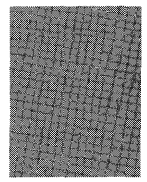
Systems Reference Library

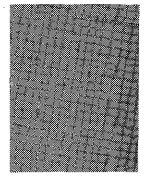
IBM System/360 Operating System

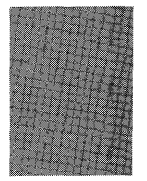
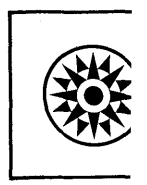
Data Management

This publication contains information concerning the data management facilities of the IBM System/360 Operating System. It provides programmers coding in the assembler language with the information necessary to design programs using these facilities.

This publication describes the cataloging, space allocation, and data access features of the operating system. Information is also included on record and label formats and data organizations.






PREFACE

This publication, primarily directed to applications programmers coding in the assembler language, is a guide to the data management facilities of the System/360 Operating System. Because it provides detailed information on the facilities available and how they are used, programmers coding in a language other than the assembler language will also find this publication useful.

This is one of a group of publications that describe the organization, functions, and applications of the System/360 Operating System. Detailed information on and coding specifications for the macro-instructions and the control statements described herein may be found in the publications IBM System/360 Operating System: Control Program Services, Form C28-6541 and IBM System/360 Operating System: Job Control Language, Form C28-6539, respectively.

Terms used in this publication are defined either in the text or in the glossary of IBM System/360 Operating System: Concepts and Facilities, Form C28-6535.

It is suggested that the reader be familiar with the information contained in the prerequisite publications listed below, as well as with the functional and operational characteristics of direct-access devices as described in the recommended publication.

PREREQUISITE PUBLICATIONS

IBM System/360 Operating System: Introduction, Form C28-6534

IBM System/360 Operating System: Concepts and Facilities, Form C28-6535

RECOMMENDED PUBLICATION

IBM 2841 Storage Control Unit, Form A24-3254

MAJOR REVISION (April, 1966)
This edition, Form C28-6537-1, obsoletes Form C28-6537-0. Significant changes have been made throughout the manual, and this new edition should be reviewed in its entirety.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

CONTENTS

ILLUSTRATIONS

FIGURES

TABLES

The data management function of the System/360 Operating System assists programmers in achieving maximum efficiency in managing the mass of data and the many programs that are processed in an installation. To attain this objective, data management facilities have been designed that provide systematic and effective means of classifying, identifying, storing, cataloging, and retrieving all data (including loadable programs) processed by the operating system.

The facilities provided can be grouped into two major categories: data set control and data access.

DATA SET CONTROL

The System/360 Operating System provides a comprehensive group of facilities that feature automatic and efficient control of many data processing operations previously performed by programming personnel as clerical tasks. A number of these control facilities are described in the following paragraphs.

Data set location control, supported by an extensive cataloging system, enables programmers to retrieve data and programs by symbolic name alone, without specifying volume serial number. In freeing computing 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 its concomitant -- human error.

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

Another major facility of the cataloging system enables programmers to classify successive generations (updates) of related data. These generations can be given similar names and subsequently be referred to relative to the current generation. The system automatically maintains a list of the most recent generations that have been produced.

Control of confidential data is a recurring managerial problem that is solved by the data set security facility of the System/360 Operating System. Using this facility, a programmer can prevent unauthorized access to payroll data sets, sales forecast data sets, and all other data sets requiring special security attention. A security protected data set is made available for processing only when the correct password is furnished.

In addition to control of data set location and security, control of direct-access storage space allocation is provided by the system. This control facility frees programmers from the details involved in allocating direct-access storage space to a data set. The programmer need specify only the amount of space required, and optionally the storage device required, and space is allocated accordingly.

The system, as well as providing a wide range of control facilities, permits the programmer to organize data sets and individual records in a variety of standard formats that can be selected to meet specific data processing needs.

DATA ACCESS

The data access facilities provided by the operating system are a major expansion of the input/output control systems (IOCS) of previous operating 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 perform the following functions:

- Read data.
- Write data.
- Block and deblock records.
- Overlap reading/writing and processing operations.
- Read and verify volume and data set labels.
- Write data set labels.
- Automatically position and reposition volumes.
- Detect error conditions and correct them when possible.

- Provide exits to user-written error and label routines.

Flexibility has been a major design principle in the system's data access facilities. The programmer and the installation can select from among eight methods of data access to obtain a group of facilities tailored to their processing requirements. Each access method supplies a comprehensive group of macro-instructions that permit the programmer to specify input/output requests with a minimum of effort; the programmer need not be concerned with learning the individual access characteristics of the many input/output devices supported by the system.

In brief, the data management facilities provided by the System/360 Operating System offer a number of advantages:

- Permit the programmer to store, modify, and refer to programs and data using the storage facilities of the system.
- Free the programmer from concern with specific input/output device configurations.
- Permit the programmer to defer such specifications as device type, block length, and buffer size until a program is submitted for execution.
- Permit the free interchange of programs and data among installations.
- Save the time and expense involved in writing routines similar to those provided.
- Allow programmers to concentrate their programming efforts on processing the records read and written by the data management function.
- Provide standardized methods for handling a wide range of input/output and related operations.
- Provide an input/output system that handles messages received from or sent to remote terminals.
- Provide the flexibility for including new or improved devices as they become available.

This section describes the data set control facilities of the System/360 Operating System. Data set identification, cataloging, security, storage space allocation, and data definition procedures are described. Information is also provided on label and record formats, and data set organizations.

DATA SETS

A data set is a named, organized collection of one or more records that are logically related. Information in data sets 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 processed by a problem program.

DATA SET NAMES

The name of a data set identifies a group of records as a data set. All data sets recognized by name alone (i.e., referable 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 DEPT999.SMITH.DATA3 is composed of three simple names that are delimited to indicate a hierarchy of categories. Starting from the left, each simple name is a category within which the next simple name is a unique subcategory.

Every 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 a data set name must not exceed 44 characters. Thus, a maximum of 22 qualification levels is possible for a data set name.

To specify the use of a particular data set by a problem program, the programmer denotes the data set name and other pertinent information (e.g., volume serial number) in a control statement called the data definition (DD) statement. (A complete description of the DD statement format can be found in the publication IBM System/360 Operating System: Job Control Language.

To permit different data sets to be processed without program reassembly, the programmer does not refer to the data set in problem programs by its name, but refers to a data control block associated with the name of the DD statement. The programmer reserves space for a data control block at assembly time by issuing a DCB macro-instruction. The DD statement is supplied in the job stream at execution time.

DATA SET CATALOGING

Keeping track of the volume on which a particular data set resides is a burden, and often a source of error. The cataloging facility of the System/360 Operating System remedies this situation. It allows the programmer to refer to data sets without specifying their physical locations. When a data set is cataloged, the serial number of its volume is associated in the catalog with the name of the data set.

THE CATALOG

The catalog of data sets is itself a data set residing on one or more direct-access volumes. It is organized into indexes that connect data set names to corresponding volume serial numbers (and to data set sequence numbers for magnetic tape volumes). Any data set residing on a direct-access or a magnetic tape volume can be cataloged.

Control Volumes

Although the complete catalog usually exists on the system residence volume, the programmer can request that parts of the catalog be placed on other volumes. Any volume, including the system residence vol-

ume, that contains part of the catalog is called a control volume. An installation can construct control volumes for cataloging specific types of functionally related data sets, such as scientific or commercial.

In general, control volumes are removable so that complete data separation may be achieved. For example, a collection of data sets might be cataloged on a removable control volume apart from the system residence volume and separate from the rest of the catalog. This volume and its related data sets could then be moved freely from one installation to another, as long as all highest level names on the control volume are entered in the catalog of the system residence volume at each installation. For any given data set, only one level of control volume other than the system residence volume is permitted.

Structure of Catalog Indexes

As previously described, a data set name consists of one or more simple names. For every distinct level of qualification in the name of a cataloged data set, the catalog includes a group of one or more blocks called an index. Catalog indexes are created and modified, as required, by a utility program. (Refer to the publication IBM System/360 Operating System: Utilities, Form C28-6586.)

The hierarchy of index levels is determined by the order of categories in the data set name. The logical records in any index are entries of the simple names and corresponding physical locations of all subordinate indexes or data sets. Every control volume contains a master index called the volume index, which has a record of the first simple name of each data set cataloged on that control volume. The lowest level index for a data set name contains the volume information needed to locate the data set itself.

Volume Indexes

A catalog search always starts in a volume index. If a specific control volume on which to begin the search has not been specified, searches of the catalog start with a scan of the volume index on the system residence volume and may continue to the volume index of other control volumes until the lowest level index for the desired data set is found.

Figure 1 represents a logical catalog structure. Every cataloged data set is cataloged under a specific hierarchy of indexes. The name of the data set specifies the indexes under which it is cataloged. The cataloged data set with the name E.A.P in Figure 1 indicates that the data set having the simple name P is cataloged under index A which, in turn, is subordinate to index E.

Figure 2 is an extension of Figure 1 showing two index hierarchies distributed between the system residence volume and another control volume. Note that index E, which is a highest level index on the control volume, has an entry in the volume index of both the system residence volume and the other control volume.

CATALOGING DATA SETS

A data set can be cataloged only if the index structure that determines its name exists in the catalog. 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 entry for an index A. The index A must have an entry directed to an index called B. When data set A.B.C is cataloged, C is entered into index B with an indication of the volume location of the data set.

Cataloging a data set with only one simple name is permissible and may be desirable if only a few data sets are to be so cataloged. These data sets can be rapidly retrieved, but retrieval time for data sets with qualified names is increased proportionally.

To catalog a data set, the programmer specifies CATLG in the DISP parameter of the DD statement. To remove references to a data set from the catalog, the programmer specifies UNCATLG in the DISP parameter. A data set can be deleted as well as have its catalog references erased, if the programmer specifies DELETE in the DISP parameter. Cataloging, uncataloging, or deletion of a data set can also be accomplished through the use of the utility program referred to in the section, "Structure of Catalog Indexes."

GENERATION DATA GROUPS

Certain data sets that are periodically updated may be chronologically related to each other. For example, similar payroll data sets may be created every week. Cataloging such data sets with unique data set names would be as inconvenient as

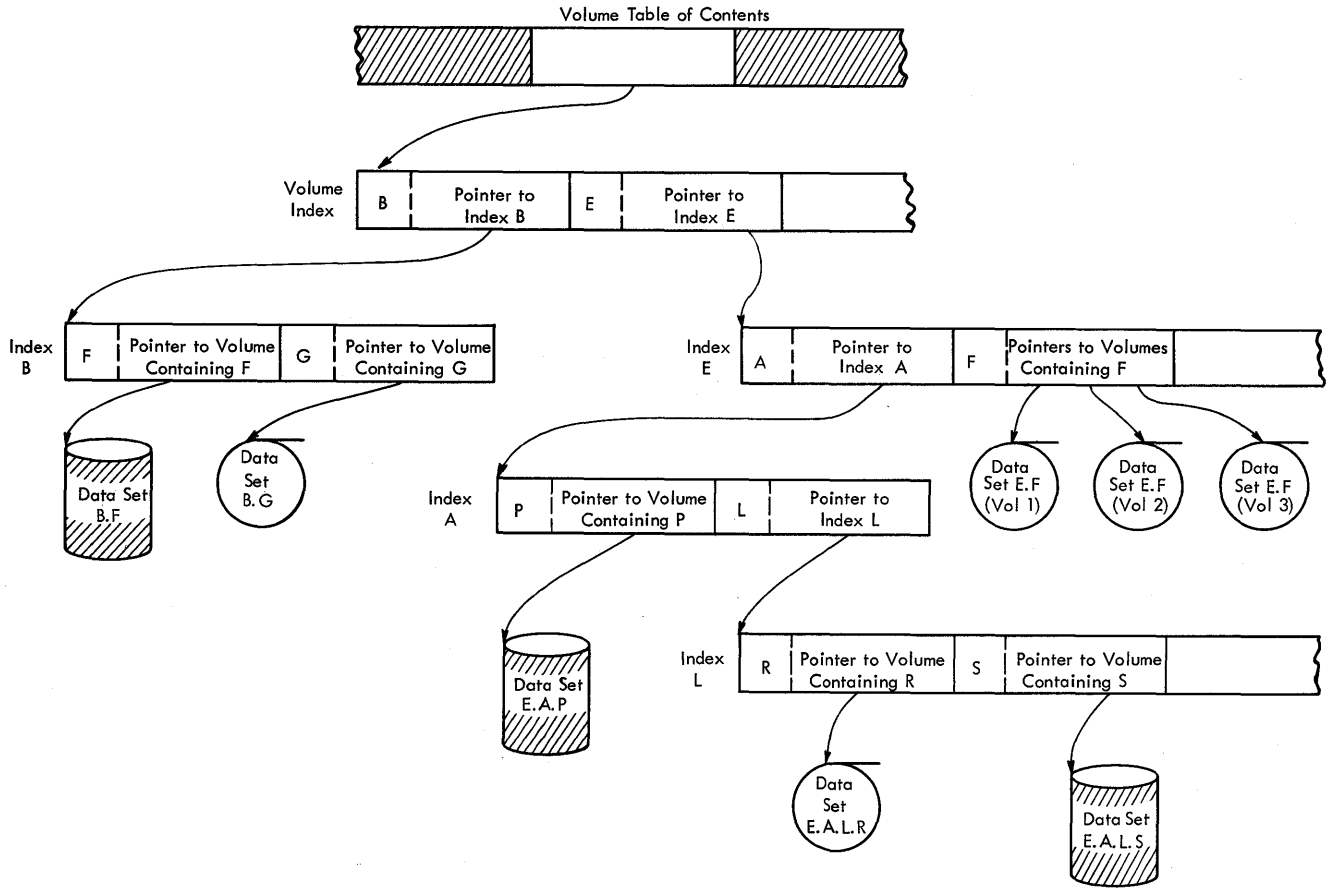


Figure 1. Format of the Catalog

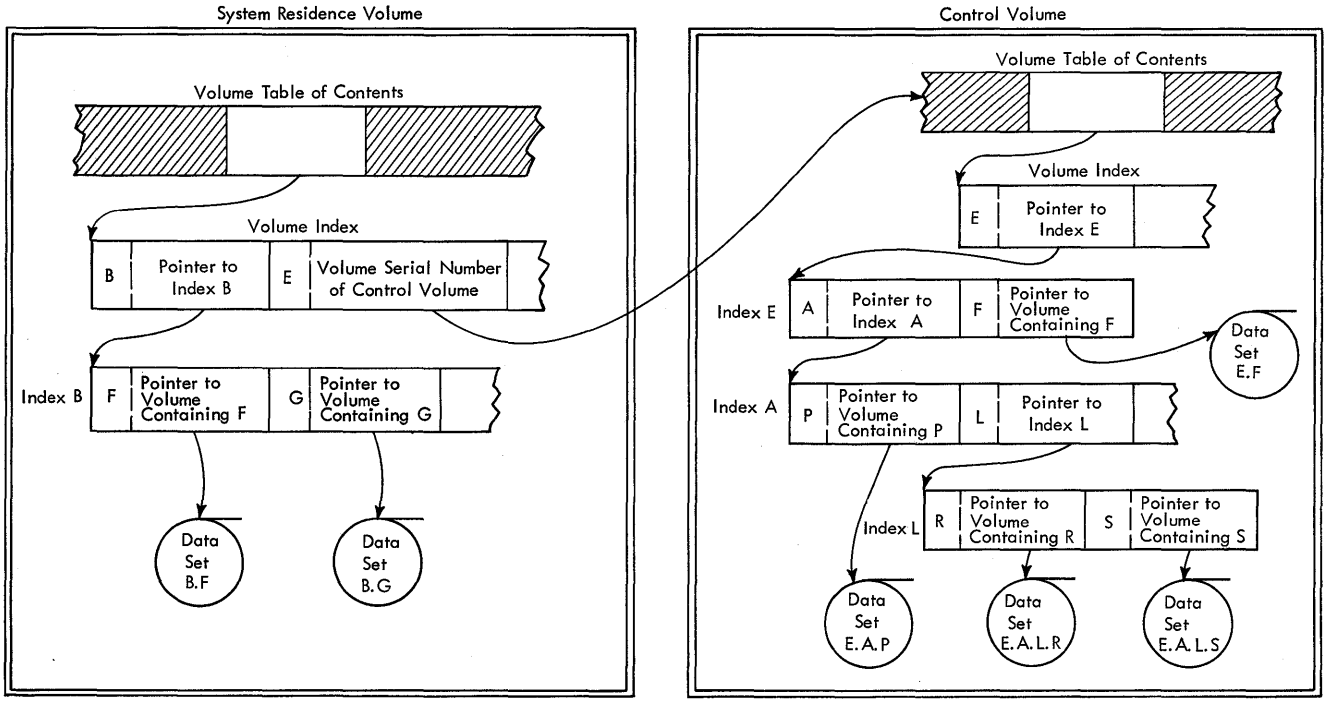


Figure 2. The Catalog on Two Volumes

giving them all the same name and accounting for volume identification. For this reason, the system provides an option with the cataloging facility that assigns numbers to individual data sets in a chronological collection, thereby allowing the programmer to catalog the entire collection under a single name. The programmer can distinguish among successive data sets in the collection without assigning a new name to each data set. Since each data set is normally created by updating the data set created on the previous run, the update is called a generation, and the number associated with it is called a generation number.

A generation data group is a collection of related cataloged data sets that can be referred to by a common name in a DD statement. The programmer can refer to a particular generation by specifying, with the common name of the group, either the relative generation number or the generation name of the data set. As explained in the following paragraphs, the relative generation number of a data set will vary in time, but its generation name is always the same.

Relative Generation Numbers

At any given time, the relative generation number of the most recently cataloged data set in any generation data group is zero. The relative generation numbers of previously cataloged data sets in the group are negative integers, indicating recency relative to that of the latest cataloged generation. New data sets for the group are created by the use of positive integers as relative generation numbers. For example, when payroll data sets compose a generation data group named PAYROLL, the programmer can refer to the most recent generation as PAYROLL(0) and to immediately preceding generations as PAYROLL(-1), PAYROLL(-2), etc. A new generation would then be referred to as PAYROLL(+1). After this new generation is cataloged, it automatically becomes PAYROLL(0), and the old PAYROLL(0) is referred to as PAYROLL(-1). Thus, adding a generation changes the relative generation numbers of all the data sets in the group.

Relative generation numbers are dependent upon the physical position of the generation name in the index. If a name is dropped from the index, the contents of the index are shifted. Therefore, if a data set from the generation data group is removed from the catalog, its relative generation number is automatically associated with the location of the immediately preceding generation.

Absolute Generation Names

To each data set in a generation data group, the system assigns an absolute generation name of the form GxxxxVyy, where xxxx is an unsigned 4-digit decimal generation number and yy is an unsigned 2-digit decimal version number. Appending the generation name to the name of the generation data group provides a unique name for the data set. For example, if 0001 is the generation number initially specified for generation data group A.PAYROLL, the programmer can refer to the first generation as A.PAYROLL.G0001V00 and to subsequent generations as A.PAYROLL.G0002V00, A.PAYROLL.G0003V00, etc.

When a data set is created by the system as a new generation, the system develops its generation and version numbers by adding the positive relative generation number specified in the DD statement to the previous generation number and setting the version number to zero. Thus, if the present generation is G1384V03 and the incrementing factor specified is the relative generation number (+2), the new generation is G1386V00. The system does not automatically create nonzero version numbers. If the programmer wishes to replace an existing generation and, optionally, to change the version number, he must specify CATLG in the DISP parameter of the DD statement for the new data set. The system automatically changes the catalog entry for the named generation. Thus, if a new data set that replaces the generation named G1386V00 is to be cataloged, it may be named G1386V01. G1386V01 replaces G1386V00 in the generation data group index when the system changes the catalog entry. The data set that was replaced is not scratched automatically.

By specifying the name of the generation data group without any generation number, e.g., A.PAYROLL, the programmer can refer to a 'concatenation' of all the existing generations. This allows all the generations currently cataloged to be retrieved as a single data set starting with the most recent generation and ending with the oldest.

CATALOGING GENERATION DATA GROUPS

In the catalog structure, a generation data group is represented by a lowest level index that contains an entry for each generation within the group. To build an index for a generation data group, the programmer specifies, for a model generation, several parameters in the BLDG state-

ment, a utility program control statement. (For a description of the BLDG statement, refer to the publication IBM System/360 Operating System: Utilities.)

One of these parameters specifies the number of generations the system is to maintain. For example, if PAYROLL(-2), PAYROLL(-1), and PAYROLL(0) are to be maintained, the programmer specifies three entries. When the specified number of entries fills the index and another generation is to be cataloged, the oldest generation is usually dropped from the generation data group index to allow space for the newest.

As an alternative, the programmer may specify a parameter that causes all the old generations to be dropped so that the newest data set effectively begins another generation data group. For example, PAYROLL generations accumulated Monday through Friday might be dropped the following Monday. The index of this generation data group is large enough for only five entries, and an attempt to catalog a sixth entry causes the index to be emptied. The sixth entry becomes the first entry of the new generation data group.

If there are exceptions to such a weekly schedule, such as holidays when no PAYROLL generation is created, the correct entries in the index are not filled unless appropriate "makeup" generations are cataloged. On Monday following a week with a holiday, the programmer could create a dummy generation to make up for the absent generation before cataloging Monday's generation. To create and catalog this extra generation, the programmer could specify the generation name G0000V00 in the DD statement and then execute a job step to fill the index with this entry.

Another parameter in the BLDG statement may be used to indicate that data sets residing on direct-access volumes are to be scratched when their entries are removed from the catalog.

DATA SET SECURITY PROTECTION

The data set security facility provides protection of data sets residing on standard labeled tapes or direct-access volumes from unauthorized use by problem programs. A protected data set cannot be made available to a problem program until a password associated with the data set is entered into the system. If the correct password is issued, the problem program can either read, write, or read and/or write the protected data set. Each protected data set has at least one entry in a cataloged data set named PASSWORD that the programmer creates on the system residence volume. Each entry consists of a 52-byte key (a 44-byte data set name field and an 8-byte password field) followed by an 80-byte record. If the name of the data set is less than 44 bytes, it must be left-justified in its field, with blanks filling all unused positions. The first two bytes of the 80-byte record contain a binary counter, which will be incremented by one every time the data set is successfully opened. The third byte of the 80-byte record is a mode field, which indicates that the problem program can either read, write, or read and/or write the data set. The remaining 77 bytes can be used at the discretion of the installation.

The PASSWORD data set itself can be protected when its own name is associated with a password (called the master password) in one of its records. This data set is created and maintained by the programmer.

When a problem program attempts to process a protected data set, the operator receives a console message requesting the password of the data set. If the operator issues the correct password (he is given two tries), the problem program is permitted to access the data set; otherwise, the program terminates abnormally, and the operator is so informed.

DATA SET STORAGE AND VOLUMES

System/360 provides an unprecedented variety of devices for collecting, storing, and distributing data. Despite this variety, the storage units have many common characteristics. For convenience, therefore, the generic term volume is used to refer to such diverse storage media as tape reels, disk packs, data cells, and drums.

Data sets are identified to the operating system by volume serial numbers. These numbers must be supplied to the system unless the data set is cataloged or passed or unless the data set is a new output data set.

DATA STORAGE ON DIRECT-ACCESS VOLUMES

Direct-access volumes play a major role in the System/360 Operating System. These volumes are used to store not only the operating system itself, but also all load modules processed by the linkage editor. In addition, direct-access volumes are used

by many installations as the chief storage media for the vast number of data sets processed each day. The following sections describe the facilities for storage space allocation on direct-access volumes.

Volume Initialization

Before a direct-access volume can be used for data storage, it must be initialized by a volume initialization utility program. This program writes standard home address and track descriptor records, checks for bad tracks, automatically assigns alternate tracks (if necessary), and initializes the remainder of each track to binary zeros. It also writes the initial volume label, provides space for additional volume labels, and writes the volume table of contents (VTOC), which is discussed in the following section. This utility program and the procedure for executing it are described in the publication IBM System/360 Operating System: Utilities.

VOLUME TABLE OF CONTENTS: The volume table of contents (VTOC) describes the contents of a 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 that 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 on the volume of the data set; its size depends on the organization and number of noncontiguous areas of the data set.

Storing a Data Set

When a data set is to be stored on a direct-access volume, the programmer must supply the operating system with control information designating the amount of space to be allocated to the data set. This information is specified by using either the SPACE, SPLIT, or SUBALLOC parameter of the DD statement for that data set. The amount of space can be specified in terms of blocks, tracks, or cylinders. Space can be allocated in a device-independent manner if the request is in terms of blocks. If the request is in terms of tracks or cylinders, the programmer must be aware of such device considerations as cylinder capacity.

ALLOCATION BY BLOCKS: When the amount of space required is expressed in terms of blocks, the programmer specifies the number, average block length, and key length of the blocks within the data set. From this information, the operating system calculates and allocates the number of tracks required. Space is always allocated in whole track units. The programmer may also request that the space allocated for a specified number of blocks begin and end on cylinder boundaries (ROUND subparameter of the SPACE parameter).

ALLOCATION BY TRACKS OR CYLINDERS: When the amount of space required is expressed in terms of tracks or cylinders, the programmer must also specify a device type in the UNIT parameter of the DD statement.

SPLIT CYLINDER: The SPLIT parameter of the DD statement permits the programmer to request that cylinder space be allocated so that portions of two or more data sets use tracks within each cylinder. For example, three cylinders might be allocated for two data sets, one data set to occupy the first two tracks of each cylinder, and the other to occupy the remaining tracks. This allocation of split cylinders reduces access time in specialized applications.

Method of Allocation: The operating system checks the VTOC to determine available areas and then allocates space in accordance with the following general rules (unless the programmer has specified otherwise):

1. If the volume on which the data set is to be stored contains more than one available area, space is allocated from the smallest available area large enough to satisfy the request.
2. If the volume on which the data set is to be stored contains no single available area large enough to satisfy the allocation request, space is allocated from more than one area. The largest available areas are used.
3. If the five largest available areas do not satisfy the allocation request, the selected volume is not used; allocation is made from another volume assigned to the data set. The programmer normally need not be concerned that noncontiguous areas on a volume

are allocated to a data set. The operating system automatically handles these discontinuities in data storage and retrieval operations.

ALLOCATION BY ABSOLUTE ADDRESS: The ABSTR subparameter of the SPACE parameter allows the programmer to request that the space to be allocated begin at a specified track address. This subparameter must be used if the volume from which space is to be allocated was initialized by an IBM System/360 Basic Operating System utility program or if any space on the volume was allocated or scratched by the basic operating system. It should also be used if the data set for which space is to be allocated contains location-dependent information in the form of absolute track addresses.

OTHER SPACE ALLOCATION OPTIONS: The DD statement provides the programmer with much flexibility in specifying space requirements. He can specify that space is to be contiguous (CONTIG subparameter) and he can suballocate space that already is assigned (SUBALLOC parameter). The programmer can also reserve either the largest single block of available space on a volume (MXIG subparameter) or up to five of the largest blocks (ALX subparameter).

Direct-Access Volume Options

The following sections describe two direct-access volume options.

TRACK OVERFLOW: If the record overflow feature is included in the computing system, the programmer can minimize unused track space on volumes by specifying the track overflow option in the DCB macro-instruction or DD statement for a given data set. When this option is used, a record that does not fit on a track is partially written on that track and continued on the next track. If this option is not used, records are not split between tracks.

WRITE VALIDITY CHECK: If the programmer specifies the write validity check option in either the DCB macro-instruction or the DD statement, the system reads each record back (without data transfer) and, by testing for a data check condition from the I/O device, verifies that each record transferred from main to direct-access storage was written correctly. This verification requires an additional revolution for each record that is written. Standard recovery procedures are initiated if an error is detected.

DATA STORAGE ON MAGNETIC TAPE VOLUMES

Because of the serial nature of magnetic tape devices, the operating system does not provide space allocation facilities comparable to those for direct-access volumes. When a new data set is to be placed on a magnetic tape volume, the programmer should specify the data set sequence number. For a data set with standard labels or no labels, the operating system positions the volume 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 the programmer has 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. The first such volume is assigned the serial number LGL001; the second, LGL002, etc.

If the programmer has specified volume serial numbers for the unlabeled volumes on which a data set is to be stored, but additional volumes are found to be required, the system assigns volume serial numbers in the same way. If, for example, the programmer has specified volume serial numbers for two volumes, but a third volume is required, it is assigned LGL003 as its serial number; if still a fourth volume is required, it is assigned LGL004; etc.

Volume serial numbers assigned by the system are of little concern to the programmer, unless data sets residing on such volumes are cataloged or passed. In this event, it is possible for data sets residing on different volumes to be cataloged or passed under identical volume serial numbers. Later retrieval of such data sets could result in unpredictable errors. For this reason, if a data set to be stored on an unlabeled tape volume(s) is to be cataloged or passed, the programmer should specify the volume serial number(s) of the volume(s) required.

VOLUME LABELING

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 are supported with standard labels only. Standard label support includes the following label groups:

- A volume label group.
- A data set label group for each data set.
- Optional user label groups, i.e., user header and/or trailer labels for each data set, for physical sequential data set organizations.

Specific information on the contents and formats of the System/360 Operating System standard labels is contained in the publication IBM System/360 Operating System: Control Program Services.

Magnetic Tape Labels

The type(s) of label processing to be supported by an installation is selected during the system generation process. Thereafter, the programmer specifies, in the DD statement, the type of label processing he is going to use for a tape data set. If he does not specify a type of label processing, standard label processing is assumed by the system.

STANDARD TAPE LABELS: Standard tape labels are 80-character records. All labels are recorded in the Extended Binary Coded Decimal Interchange Code (EBCDIC), with the exception of 7-track tape labels, which are recorded in Binary Coded Decimal (BCD). The density of a tape label is the same as that of the data on the tape, which was specified in the DD statement or the DCB macro-instruction.

The organization of standard tape labels for a single tape volume and one data set is illustrated in Figure 3.

Volume Label Group: The volume label group consists of an initial volume label, which is created by a utility program, and a maximum of seven additional volume labels. These additional labels are processed by means of an installation routine that is incorporated into the system.

In the System/360 Operating System, a volume label identifies a volume and its owner. Although primarily intended to verify that the correct volume is mounted, volume labels also permit volume protection by preventing use of the volume by unauthorized programs.

A tape using standard tape labels is identified as such by the operating system when it reads the initial record and determines that it is an initial volume label by finding that the first four characters of the record are VOL1 (volume label 1).

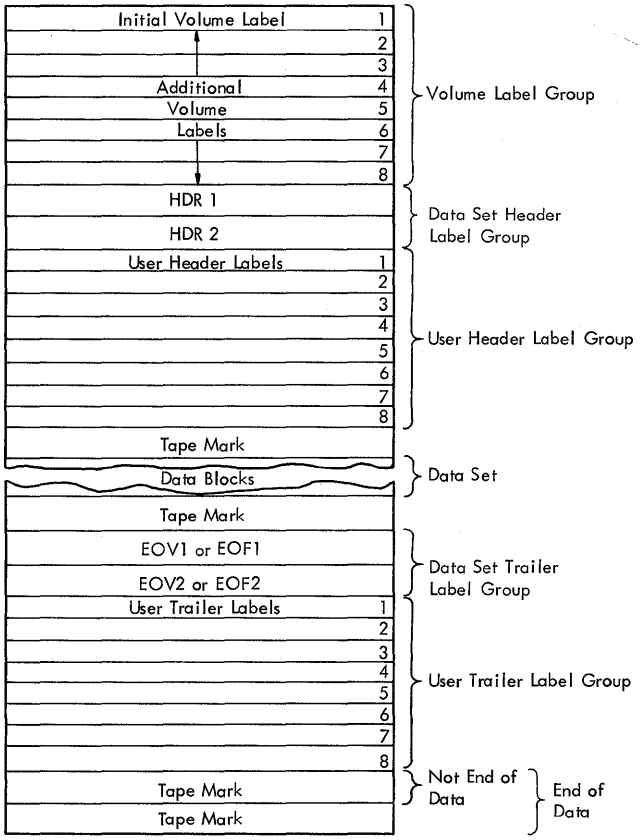


Figure 3. Organization of Standard Tape Labels

Data Set Header Label Group: The data set header label group consists of two data set header labels: HDR1 and HDR2. HDR1 contains operating system data and device-dependent information; HDR2 contains data set characteristics. (Additional data set header labels are not supported.) These labels are created by the operating system in accordance with a fixed format when the data set is recorded on tape. They can then be used to locate the data set, to verify references to the data set, and to protect it from unauthorized use.

User Header Label Group: Optionally, a maximum of eight user header labels can appear on tape immediately following the data set header labels. The operating system writes these labels as directed by the problem program recording the tape. The first four characters of the user header label must be UHL1,...,UHL8; the remaining 76 characters can be specified by

the user. When the tape is read, the operating system makes the user header labels available to the problem program.

Data Set Trailer Label Group: The data set trailer label group consists of two trailer labels: EOV1 or EOF1 and EOV2 or EOF2. These labels are identical to the data set header labels except that:

- The label identifier EOV1 or EOF1 replaces HDR1, and the label identifier EOV2 or EOF2 replaces HDR2.
- The first label (EOV1 or EOF1) contains a block count field that indicates the number of blocks of the data set recorded on the tape. It is used for checking purposes. This field contains zeros in the HDR1 label.

These labels duplicate the data set header labels to facilitate backward reading of the tape. Location, verification, and protection of the data sets can also be achieved with data set trailer labels.

User Trailer Label Group: A maximum of eight user trailer labels can immediately follow the data set trailer labels. 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.

NONSTANDARD TAPE LABELS: When nonstandard labels are used for magnetic tape, nonstandard label processing routines are supplied by the installation and incorporated in the operating system at system generation time. Detailed information about writing nonstandard label processing routines is contained in the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550.

The number and contents of nonstandard labels can be specified by the user, except that the initial record cannot be a standard tape volume label; i.e., the first four characters of this record cannot be VOL1. When these first four characters are not VOL1, the operating system transfers control to the nonstandard label processing routines. These routines provide volume positioning that is compatible with the positioning techniques used by the system's standard label processing routines. The operating system assumes that a tape using nonstandard labels is properly positioned upon completion of a nonstandard label processing routine.

Magnetic Tape Volume Organization

If a data set on magnetic tape is to be processed by an IBM data access method, the data set must be preceded and followed by a tape mark, with one exception: the first data set on an unlabeled tape volume is not preceded by a tape mark. (Note that if a tape mark should precede the first data set on an unlabeled tape volume, the data set sequence number of that first data set is two.) Tape marks may not exist within a data set.

When an access method is used to create a tape data set with standard labels or no labels, all tape marks are automatically written by the system. If standard label processing is being used, one tape mark follows each trailer label group, with one exception: two tape marks are written after the last trailer label group on a volume when it indicates end of data set (EOF). If an unlabeled volume is being used, an additional tape mark is written after the tape mark that follows the last data set on the volume.

When an access method is used to create a tape data set with nonstandard labels, neither of the delimiting tape marks is written by the system. If an access method is to be used to retrieve the data set, these tape marks should be written by the appropriate installation nonstandard label processing routine. The system does not require that one or more tape marks be written after nonstandard trailer labels, since installation nonstandard label processing routines must handle the positioning of tape volumes.

Figure 4 illustrates the organization of standard labels and data on magnetic tape for the following tape organizations.

Single Data Set/Single Volume: Volume labels are followed by data set header labels and optional user header labels. The data set is preceded and followed by a tape mark. The data set trailer labels (EOF) appear after the tape mark following the data set and are followed by the optional user trailer labels. Two tape marks are the last elements of this tape organization.

Single Data Set/Multiple Volumes: This class of tape organization is an expansion of the previous class. More than one volume is necessary to contain the data set. The last volume is organized the same as the single data set/single volume class. All other volumes are also organized in this way with the exception of their data set trailer label groups. The EOV trailer group appears in place of the EOF trailer

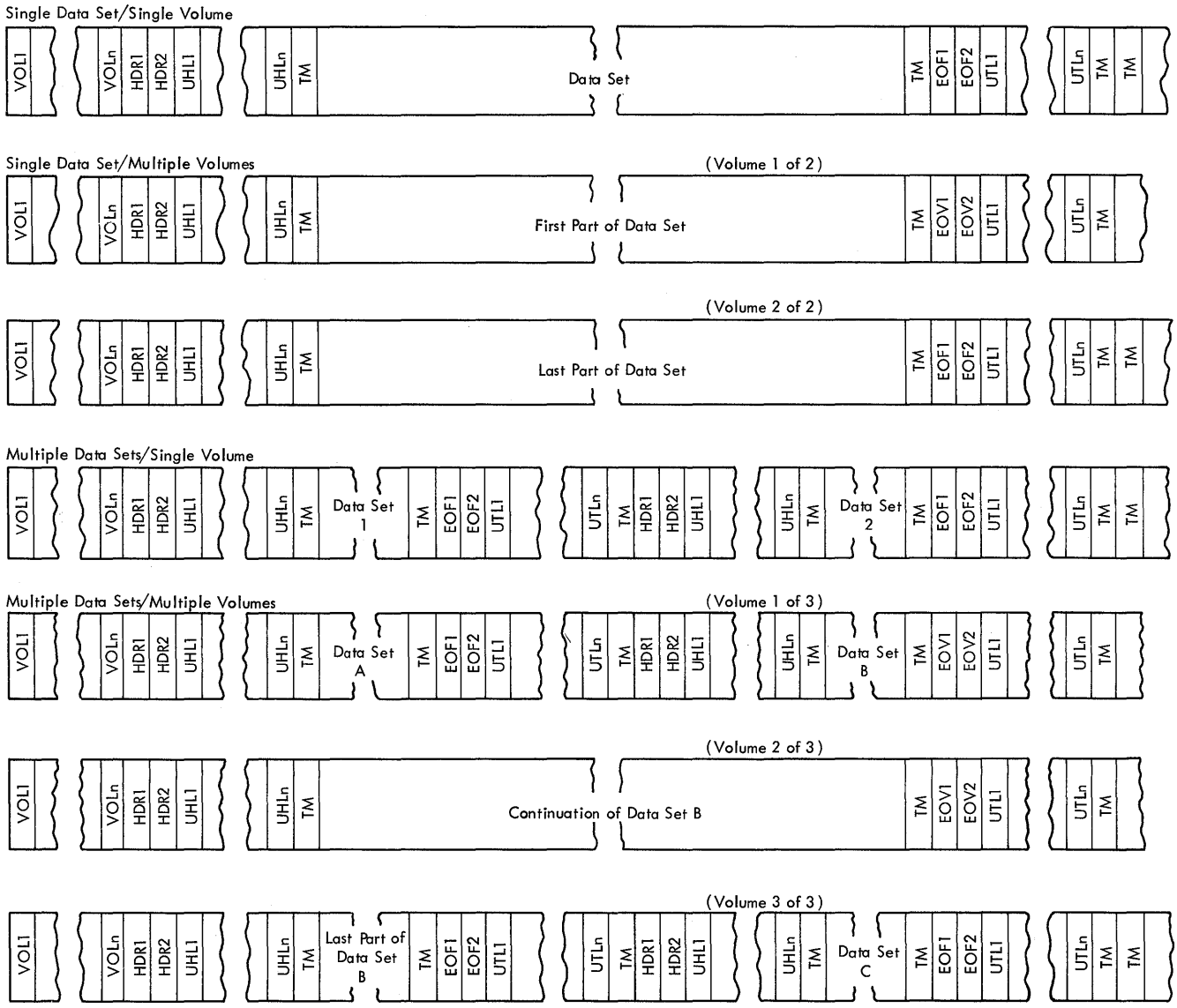


Figure 4. Standard Label Formats for Magnetic Tape

set. The EOV trailer group is followed by one tape mark. The data set and user labels are repeated on each volume, but the volume labels differ on each tape.

Multiple Data Sets/Single Volume: The volume begins with a volume label group. Each data set starts with a data set header label group, followed by the optional user header label group, and a tape mark. The data set then appears, followed by a tape mark, an end-of-file trailer label group, a user trailer label group, and another tape mark. The last data set on the volume differs only in that the user trailer labels are followed by two tape marks.

Multiple Data Sets/Multiple Volumes: This class of tape organization is similar to the previous one. However, more than one volume is required because of the amount of information. The end-of-volume trailer labels are identical to those of the single data set/multiple volumes tape organization.

Direct-Access Labels

Only standard label formats are used on direct-access volumes. Volume, data set, and optional user labels are used (see Figure 5). In the case of direct-access volumes, the data set label group is the data set control block (DSCB).

VOLUME LABEL GROUP: The volume label group immediately follows the initial program

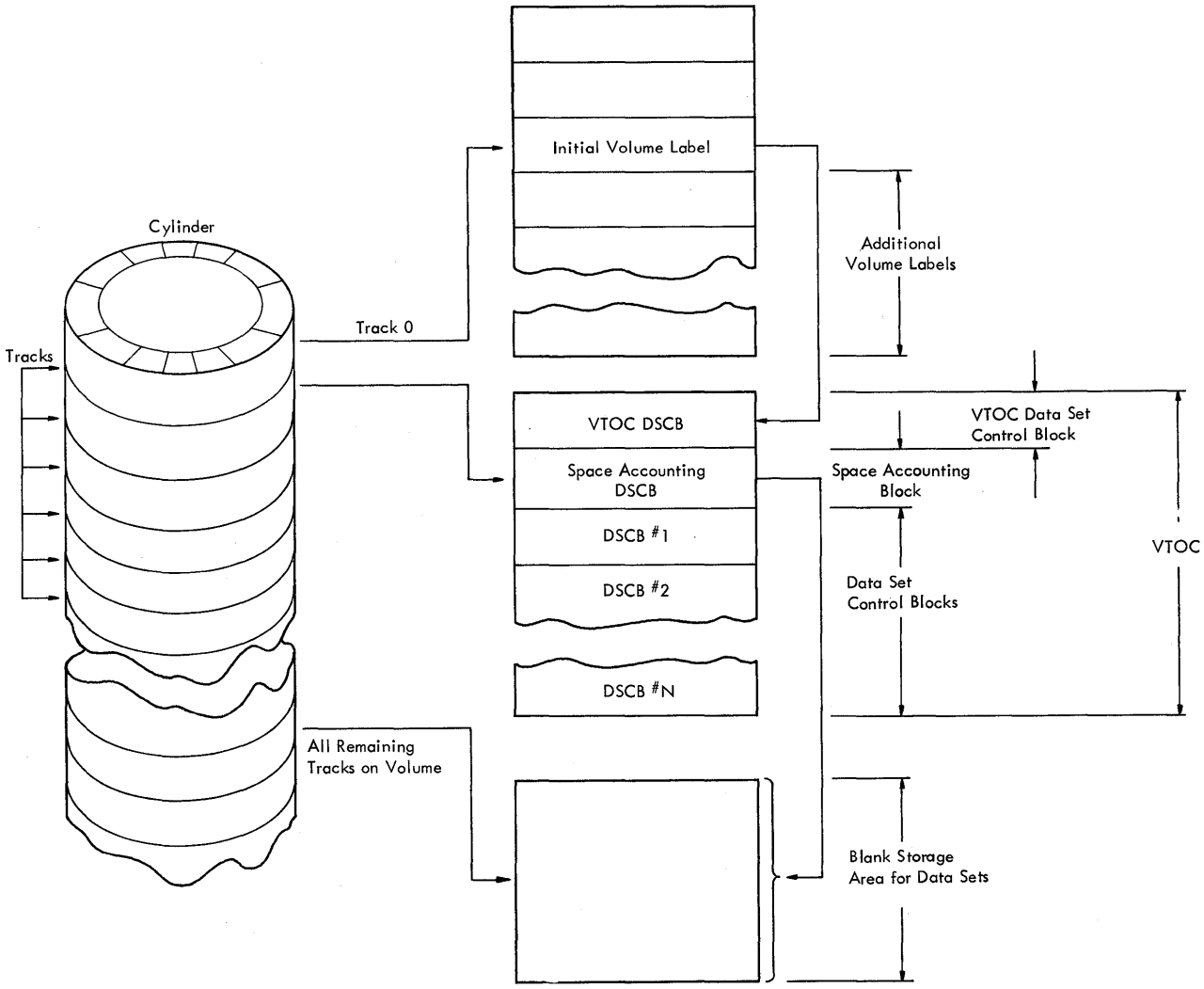


Figure 5. Direct-Access Labeling

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. These additional labels are processed by means of an installation routine that is incorporated into the system.

DATA SET CONTROL BLOCK (DSCB): The system automatically constructs a DSCB when space is requested for a data set on a direct-access volume. Each data set on a direct-access volume has a corresponding data set control block to describe its characteristics.

USER LABEL GROUP: If the LABEL parameter of the DD statement indicates that user labels are to be used, and if the programmer has requested space allocation in terms of cylinders by using CYL, SPLIT, or ROUND, the data set is automatically allocated one additional track for user labels. Otherwise, the first track allocated for the data set is reserved for user labels. The current minimum track size supports a maximum of eight 80-character user labels (including both user header and trailer labels). Consequently, a program that creates more than eight user labels becomes device-dependent among direct-access devices. If device independence is not desired, the program may create up to eight user header labels and eight user trailer labels, not to exceed one track.

DATA SET RECORD FORMATS

Data processing operations are concerned with individual data records within a data set. All records of a data set must have the same format. The formats of these records is the subject of the following discussion.

BLOCKS

The data between interrecord gaps is called a block; each block can consist of one or more logical records.

LOGICAL RECORDS

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

RECORD BLOCKING

Blocking of records, shown in Figure 6, is the process of grouping a number of logical records before writing them on a volume. Blocking improves effective data rate and conserves storage space on the volume by reducing 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.

RECORD FORMATS

Logical records may be in one of three formats: fixed-length (format F), variable-length (format V), or undefined (format U).

The prime consideration in the selection of a record format is the nature of the data set itself. The programmer knows the type of input his program will receive and the type of output it will produce. His selection of a record format is based on this knowledge, as well as an understanding of the type of input/output devices that are to handle the data set and of the access method used to read or write the data set.

The record format of a data set is placed into the data control block according to specifications in either the DCB macro-instruction, the DD statement, or the data set label.

Fixed-Length (Format F)

Format F records are fixed-length records. The number of logical records within a block (blocking factor) is normally constant for every block in the data set unless the data set contains truncated blocks (short blocks).

In unblocked format F, the logical record constitutes the block.

The system performs physical length checking on blocked format F records, automatically 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 have records in standard F format if all the records in the data set are in F format, if each track except the last is filled to capacity, and if no blocks besides the last are truncated. Standard F data sets can be read from direct-access devices more efficiently than data sets with truncated blocks, because the system can predict the location of each block to be read.

Format F records are shown in Figure 7. The optional control character (C), used for stacker selection and carriage control, may be included in each logical record to be printed or punched.

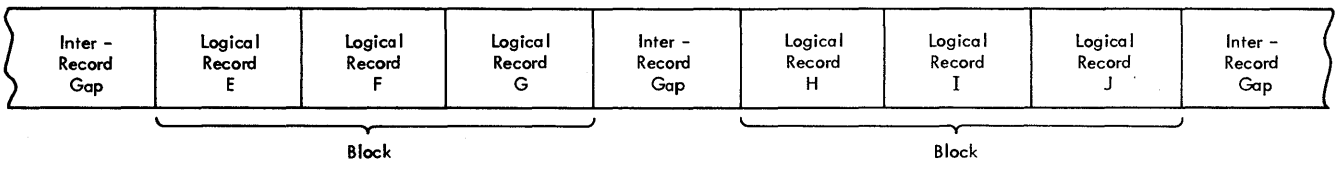


Figure 6. Blocked Logical Records

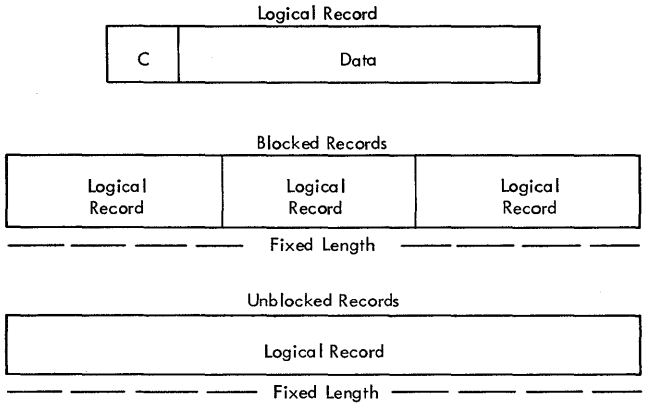


Figure 7. Format F Records

Variable-Length (Format V)

Format V provides both for variable-length records, each of which describes its own length, and for variable length blocks of such records, each of which includes a block length. The system performs length checking of the block and makes use of the record length information in deblocking and blocking. Format V records are shown in Figure 8. The first four bytes of the logical record contain control information: 'll' represents the length of the logical record and 'bb' represents the two bytes reserved for system use. These bytes must be provided by the user when he is creating the record. The optional control character (C) may be specified as the fifth byte of each logical record.

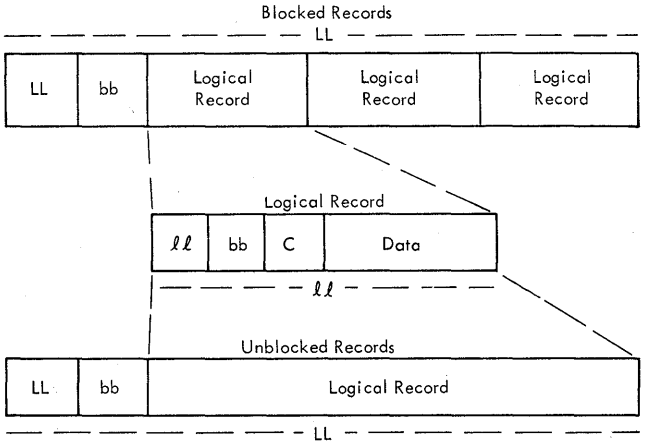


Figure 8. Format V Records

In format V, 'LL' represents the block length and 'bb' represents the two bytes reserved for system use. With the exception of the basic sequential and the basic direct access methods, these characters are automatically provided when the data set is written. Both input and output buffer areas must be large enough to accommodate the additional four bytes.

In unblocked format V, one logical record and the block control information constitute the block.

The initial eight bytes (nine if the optional control character is specified) of the block are not printed or punched. The use of these eight bytes for any purpose other than that specified in this section may cause unpredictable results.

Undefined (Format U)

Format U is provided to permit the processing of any blocks that do not conform to the F or V formats. Format U records are shown in Figure 9. The optional control character (C) may be used in each logical record.

Since each block is treated as a logical record (unblocked), any deblocking must be accomplished by the user's program. The system does not perform length checking on format U records. For this reason, a program can be designed to read less data into main storage than is actually contained in a block.

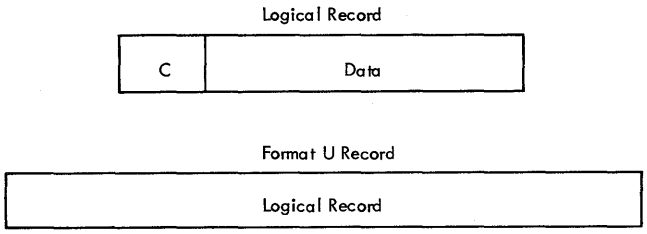


Figure 9. Format U Records

CONTROL CHARACTER

The programmer may optionally specify, in the DD statement or the DCB macro-instruction, that a control character is part of each logical record in a data set. This character specifies carriage control when the data set is printed or stacker selection when the data set is punched. The character itself is never printed or punched but is a part of the record in storage. Buffer areas must be large enough to accommodate this character (byte). If the immediate destination of the record is

a device that does not recognize this control character, e.g., disk, the system assumes that the control character is the first byte of the data.

If the destination of a record is a printer or a punch and the user has not specified that the first byte of the data is to be used as a control character, this byte is simply treated as the first byte of the data.

DATA SET ORGANIZATIONS

Data set organization refers to the physical arrangement of data set records. To give the programmer maximum flexibility and efficiency in reading and writing data sets, the operating system provides five types of data set organization:

1. Sequential.
2. Partitioned.
3. Indexed sequential.
4. Direct.
5. Telecommunications.

Only sequential organization can be used for data sets on magnetic tape volumes, unit record equipment, and paper tape.

The most desirable data organization for a given data set depends on how the data set is to be used. The organization determines, to a great extent, the access methods that can be used with the data set. The programmer specifies the organization of a data set in the DCB macro-instruction. The data set organizations are described in the following sections.

SEQUENTIAL ORGANIZATION

A sequential data set is one whose records are organized solely on the basis of their successive physical positions in the data set. As indicated in Figure 10, the records exist sequentially within the volume space allocated to the data set and are read or written in the same order in which they appear. For example, the fourth record of the data set in Figure 10 normally is read only after the first three records have been read. If the data set is stored on more than one volume, each volume must be of the same device type. Automatic volume switching is provided by the system. Individual records cannot be deleted or inserted unless the entire data set is rewritten. This organization is generally used for a data set most of whose records are processed each time the data set is used.

Space Allocation for Sequential Data Sets

The programmer must request space for sequentially organized data sets that are to be written on direct-access volumes. If the programmer is uncertain of the amount of space required by a particular data set, he can ensure sufficient space by specifying the secondary quantity subparameter of the SPACE parameter of the DD statement. This subparameter indicates an amount of space to be allocated if the original space allocated to the data set is exhausted and more data is to be written. This additional space is allocated as required from the volume(s) assigned to the data set. The programmer can also release any space that he does not use by specifying the RLSE subparameter in the SPACE parameter.

PARTITIONED ORGANIZATION

A partitioned data set is one that is divided (partitioned) into sequentially organized members made up of one or more records. All members have identical properties, such as record formats, block length, options selected, etc. Each member of a partitioned data set has a unique simple name that does not exceed eight characters in length. Records of any given member are retrieved or stored successively according to physical sequence.

The main advantage of a partitioned data set is that it gives the programmer the

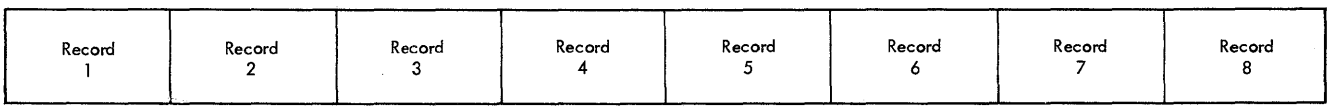


Figure 10. A Sequentially Organized Data Set

ability to retrieve individual members, once a single data control block is opened. For example, a library of subroutines might be a partitioned data set whose members are the subroutines; within each subroutine, records are sequentially organized. Individual members may be added or deleted as required. A partitioned data set must reside on one direct-access volume.

Directory

Each member of a partitioned data set is associated with an entry in a series of records at the beginning of the data set called a directory. As indicated in Figure 11, the directory contains the name, and corresponding position within the data set, of every existing member. Optionally, the programmer can place up to 62 characters of additional information into an entry. Directory entries are maintained in order of collating sequence of member names.

To provide location independence, the system records the track address of a member as a relative track within the data set rather than as an absolute track address. This allows an entire data set to be moved without a change in relative track addresses. The first track of a data set is relative track 0, the second is relative track 1, etc. Thus, the data set can be considered as one continuous set of data tracks regardless of how the space has actually been allocated.

Space Allocation for Partitioned Data Sets

When the programmer requests the amount of space to be allocated to a partitioned data set, he also specifies the amount of space required for the directory. If the storage space specified by the programmer for the members is filled, additional space for new members is dynamically allocated by specifying the secondary quantity subparameter of the SPACE parameter of the DD statement. The programmer can release any space that he does not use by specifying the RLSE subparameter. If the space for the directory is filled, however, no new members can be added to this data set. Once allocated, space for the directory cannot be expanded. The data area occupied by deleted or replaced members of a partitioned data set is not reusable until a COPY PDS statement is executed to copy the remaining members of the data set onto the same or a different volume. The COPY PDS statement, a utility program control statement, is described in the publication IBM System/360 Operating System: Utilities.

INDEXED SEQUENTIAL ORGANIZATION

An indexed sequential data set is one whose records are organized on the basis of a collating sequence determined by keys that precede each block of data. The key for each block of data is 1-255 bytes in length and is identical to the key of the

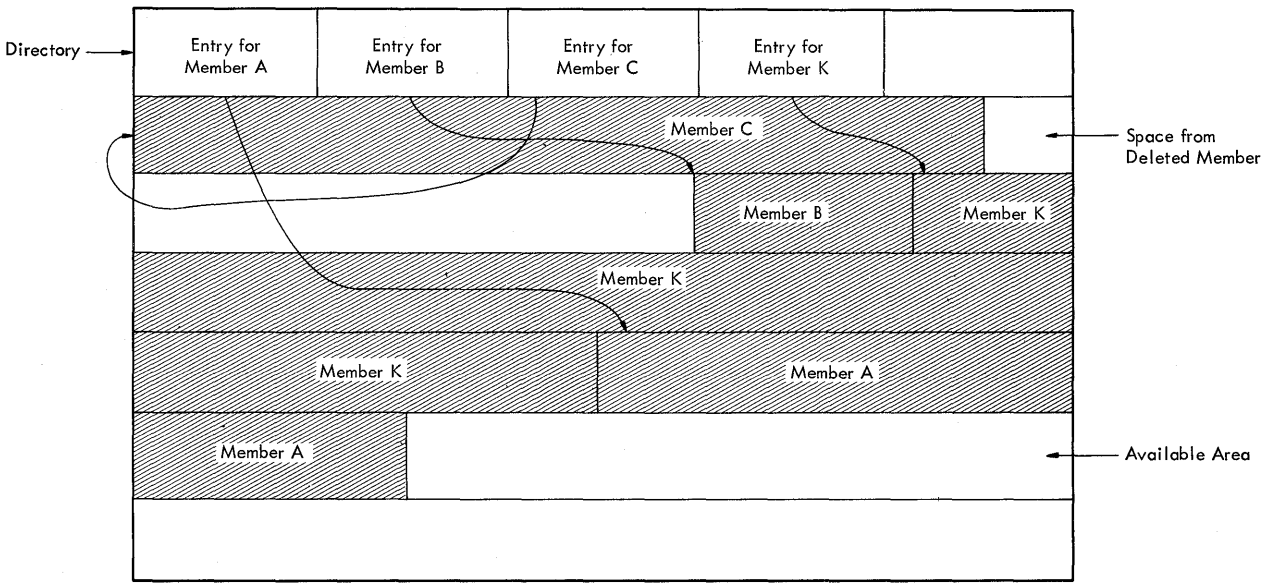


Figure 11. A Partitioned Data Set With Four Members

last record in that block. An indexed sequential data set exists in space allocated on direct-access volumes as prime areas, overflow areas, and indexes.

Indexed sequential organization gives the programmer much flexibility in the operations he can perform on a data set. He has the ability to:

- Read or write (in a manner similar to that for sequential organization) logical records whose keys are in ascending collating sequence.
- Read or write individual logical records whose keys are in any order. Reading records in this manner is somewhat slower per record than reading according to a collating sequence, because a search for pointers in indexes is required for the retrieval of each record.
- Add logical records with new keys. The system locates the proper position in the data set for the new record and makes all necessary adjustments to the indexes.

Indexes

The ability to read and write records from anywhere in a data set with indexed sequential organization is provided by indexes that are part of the data set. There are two types of indexes: a cylinder index for the whole data set and a track index for each cylinder. An entry in a cylinder or track index contains the identification of a specific cylinder or track and the highest key that is on that cylinder or track. The system locates a given record by its key after a search of a cylinder index and a track index within that cylinder.

Master Index

If a data set occupies many cylinders, a serial search of the cylinder index for a key is inefficient. The programmer can cause a master index to be created that indexes the cylinder index, as shown in Figure 12. A master index will be constructed only if the cylinder index occupies more than one track. The programmer specifies in fields of the data control block that, if the size of a cylinder index exceeds a certain number of tracks, a master index should be created. Each entry in the master index points to a track of the cylinder index. If the size of the master index exceeds the number of tracks specified in the data control block, the master index is automatically indexed by a higher level master index. Three such higher level master indexes can be constructed.

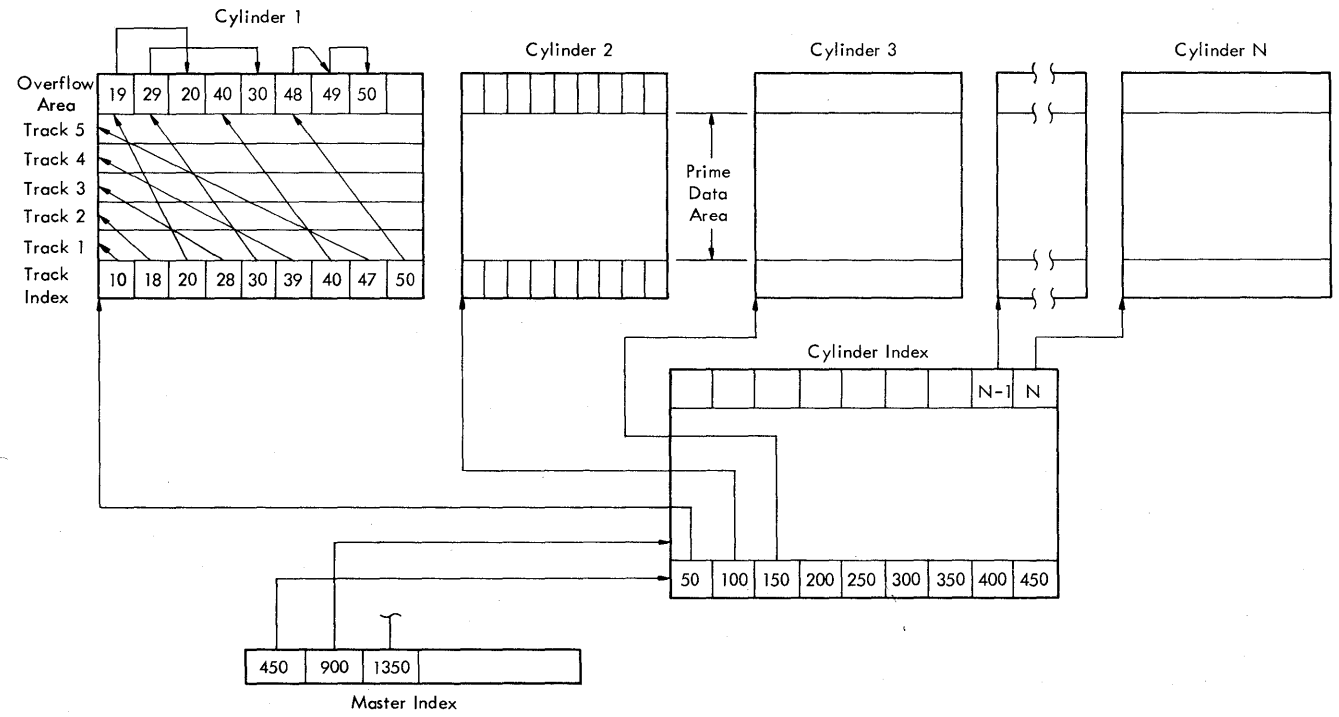


Figure 12. Index Structure for an Indexed Sequential Data Set

Insertion of Records

A new record added to an indexed sequential data set is placed into a location on a track determined by the value of its key field. If records were inserted in precise physical sequence, insertion would require shifting all records of the data set with keys higher than that of the one inserted. However, because an overflow area exists, indexed sequential data organization allows a record to be inserted into its proper position with only the records on the track in which the insertion is made being shifted.

Overflow Area

In addition to the prime area, whose tracks initially receive records of an indexed sequential data set, there is an overflow area for records forced off their original tracks by the insertion of new records. When a record is to be inserted, the records already on the track that are to follow the new record are written back on the track after the new record. If the addition of records results in insufficient track space for all the records to be written onto the track, the records that do not fit are written onto an overflow track. Track index entries are adjusted to indicate records on an overflow track. Figure 13 illustrates this adjustment for addition of records to an indexed sequential data set whose keys are in a numerical collating sequence.

When this data set is created, its records are placed on the prime tracks in the storage area allocated to the data set. If a record, e.g., record 7, is to be inserted into the data set, the indexes indicate that record 7 belongs on prime track 1. Record 7 is written immediately following record 5, and records 8 and 10 are retained on prime track 1. Since record 11 no longer fits on this track, it is written on an overflow track and the proper track index is adjusted to show that the highest key on prime track 1 is 10 and that overflow records exist. When records 17 through 22 are added, prime track 2 receives records 17 to 21, but record 22 does not fit and is written following record 11 on the overflow track. When record 9 is inserted, record 10 is shifted to the overflow track after record 22. Note that records 10 and 11 on the overflow track are chained together to show their logical sequence and to indicate that they belong on the same prime track.

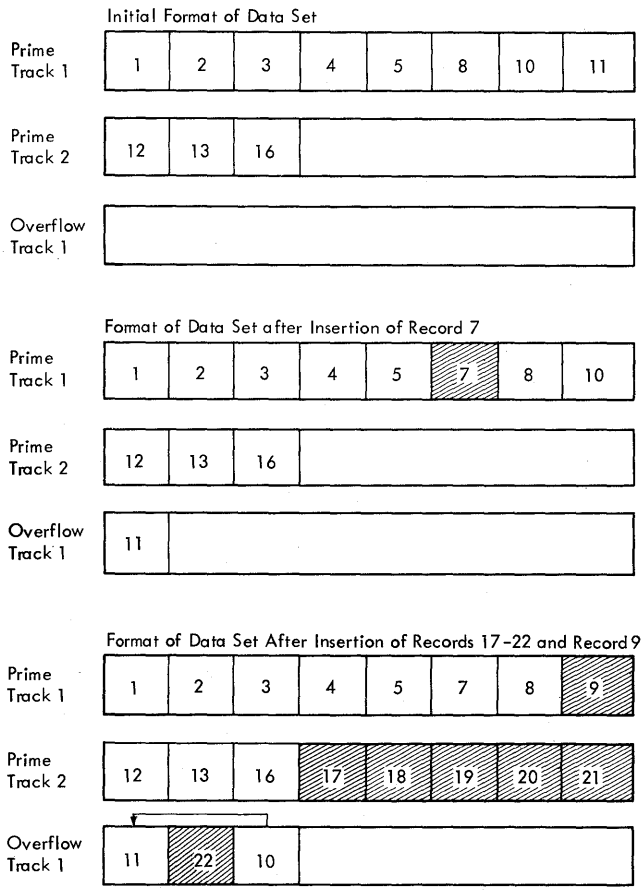


Figure 13. Addition of Records to a 1-Cylinder, 3-Track Indexed Sequential Data Set

Two types of overflow areas may be requested by the programmer. He can request a cylinder overflow area that provides a certain number of tracks on each cylinder to hold the overflow of that cylinder. He can also request an independent overflow area that provides a certain number of tracks independent of the rest of the data set, perhaps even on a different volume.

The programmer may mark records for deletion. If a marked record is forced off its original track by the insertion of a new record, then this record is not written in an overflow area and is thereby deleted from the data set.

Space Allocation for Indexed Sequential Data Sets

Only the ABSTR or the CYL subparameter of the SPACE parameter of the DD statement can be used when allocating space for an

indexed sequential data set. All requests must be for an integral number of cylinders.

The storage area allocated to an indexed sequential data set must be sufficient to include all space required for prime, index, and overflow purposes. The procedures to be followed in allocating this space differ, depending upon the programmer's index location requirements, and his need for an independent overflow area:

- Space for the prime area can be allocated on more than one volume, but each volume must be of the same device type.
- Space for a separate index area can be allocated on a volume and device type different from that of the prime area, if the prime area is on an IBM 2321 Data Cell Drive.
- Space for an independent overflow area can be allocated on a separate volume of the same device type as that of the prime area.

Basically, there are four space allocation techniques, each causing the index area to be allocated differently. Note that for those techniques requiring more than one DD statement, only the first statement may contain a symbol in its name field. The programmer must indicate the type of area for which space is to be allocated by specifying either (INDEX), (PRIME), or (OVFLOW) as the element portion of the DSNAME parameter of the DD statement. If no element name is provided, (PRIME) is assumed. If a DD statement specifying (INDEX) is used, it must precede a DD statement specifying (PRIME). If a DD statement specifying (OVFLOW) is used, it must follow a DD statement specifying (PRIME).

One technique for allocating space is to prepare a separate DD statement for each type of area required. If in addition to the prime area, for example, the programmer requires a separate index area and an independent overflow area, three DD statements should be prepared.

If the programmer does not require a separate index area, he can cause the index area to be embedded within, or placed at the end of, the prime area, or to be placed at the end of an independent overflow area using one of the following three techniques for allocating space:

- To develop an index area embedded within the prime area (in order to reduce access arm movement during the processing of the data set), the programmer should prepare a DD statement for the prime area. In the directory quantity subparameter of the SPACE parameter of this statement, the programmer should specify how much of the prime area is to be used for index purposes. An additional DD statement requesting a separate index area should not be prepared. The programmer may, however, use a DD statement requesting space for an independent overflow area.
- To develop an index area at the end of the prime area, the programmer should prepare a DD statement requesting prime area space that does not contain an embedded index area. The space requested for the prime area must be large enough to include an index area. Additional DD statements requesting space for a separate index area and for an independent overflow area should not be used.
- To develop an index area as part of an independent overflow area, the programmer should prepare two DD statements: one to allocate space for a prime area that does not contain an embedded index area, and one to allocate space for an independent overflow area. The space requested for the overflow area must be large enough to include an index area. An additional DD statement requesting a separate index area should not be prepared.

DIRECT ORGANIZATION

A direct data set is one whose blocks are organized on a direct-access volume in any manner chosen by the programmer. If the data set is to be stored on more than one volume, each volume must be of the same device type. When a request to store or retrieve a block is made, either an address relative to the beginning of the data set or an actual address (i.e., device, cylinder, track, block position) must be furnished. This address can be specified as being the address of the desired block or as a starting point within the data set where the search for the block is to begin. When a block search is specified, the programmer must also furnish the data key (e.g., part number, customer name) that is associated with the desired block.

Insertion of Blocks

With direct organization, the programmer can indicate, without specifying a particu-

lar block position, a track on which a block is to be written. The block is simply written into the first available space on the track. Figure 14 illustrates a data set with format V or U blocks that have been distributed over allocated tracks according to an "item number" recorded in a key field with each block. If items 10, 21, and 50 are to be inserted into this data set and the track selection algorithm indicates that they belong on relative tracks 0, 1, and 2, respectively, these blocks are placed on the proper tracks after the existing blocks.

TELECOMMUNICATIONS ORGANIZATION

A description of the facilities for telecommunications is given in the publication IBM System/360 Operating System: Telecommunications, Form C28-6553.

DATA SET DEFINITION

Before a data set can be made available to a problem program, the system requires that descriptive information defining the data set be placed into a data control block. Sources of information for the data control block are the DCB macro-instruction, the DD statement, and, optionally, a data set label. The system stores information from the DD statement in a control block called the job file control block (JFCB). The JFCB is placed in a job queue on a direct-access volume when the job associated with it is scheduled for execution. The following sections discuss the data control block and the procedures followed by the programmer to define data sets.

DATA CONTROL BLOCK

A data control block is a group of contiguous fields in the user's assembled program that provide information about a data set to the system for scheduling and executing input/output operations. The fields describe the characteristics of a data set (e.g., data set organization) and its processing requirements (e.g., number of buffers required for input/output operations). By issuing a DCB macro-instruction, the programmer requests that a data control block be constructed at assembly time. Subsequently, several sources can be used to enter information into the data control block fields. The process of filling in the data control block is completed at execution time.

This information may come from the DCB macro-instruction itself, from the JFCB, or from the data set label. Only unspecified

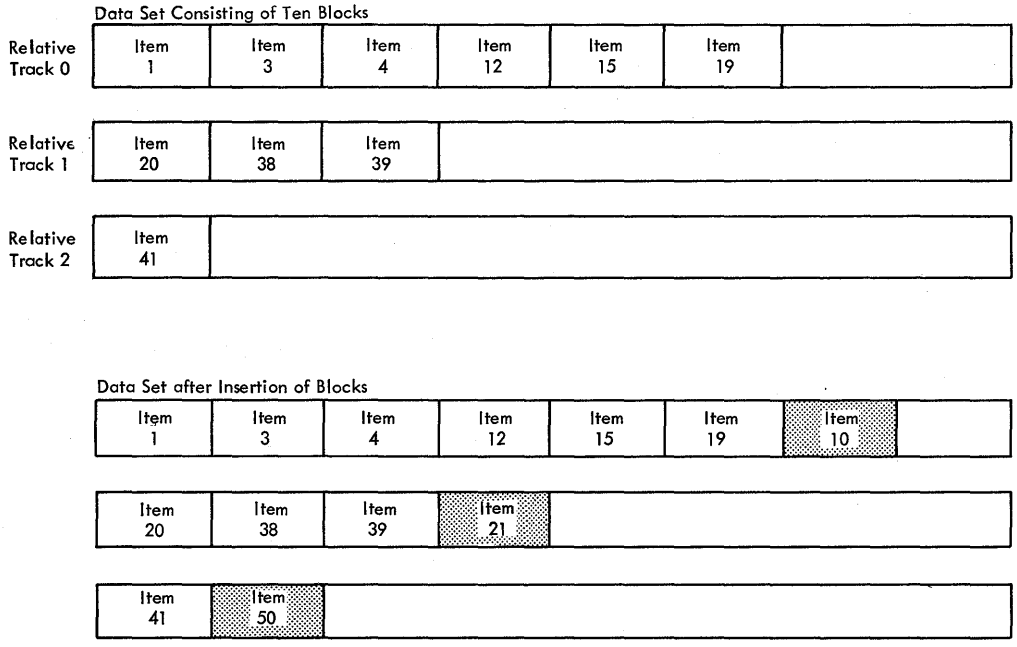


Figure 14. Addition of Blocks to a Direct Data Set

fields can be filled from each source. For example, if a field is specified in both the DD statement and the data set label, only the information for that field which is supplied by the DD statement is used for the data control block; the corresponding information in the data set label is ignored. However, the programmer can write routines that modify any data control block field. In Figure 15, lines 1 to 4 indicate the sequence in which the sources of information for the data control block are used. Solid lines indicate required sources, and dotted lines indicate optional sources.

The above transfer of information is part of a procedure called opening a data control block. Opening a data control block for a data set makes that data set available to a problem program. Closing the data control block disconnects that data set from the problem program. (Two macro-instructions, OPEN and CLOSE, are provided to perform these operations.) Thus, a completed data control block may exist only during program execution.

There is usually a one-to-one correspondence between data sets and opened data control blocks. Normally, one data control block should be opened for one particular data set. However, concurrently opening more than one data control block for the processing of the same data set is not prohibited for direct-access operations. (It is prohibited for tape operations.)

Caution: If the programmer opens more than one data control block for the same data set, he must be aware of the implications of this procedure and exercise caution with respect to such items as multivolume data sets, secondary allocation, volume positioning and the dependence of certain macro-instructions upon positioning, and label processing.

Job File Control Block

At the time a data control block is opened, information is transferred into the JFCB. The portion of this transfer of information that pertains to completing the data control block is illustrated by line 2 in Figure 15. After the data control block has been completed, further modification of the JFCB is possible as illustrated by lines 4 and 5 in Figure 15. Once the JFCB has been modified, it remains so until it is no longer needed by the system.

Data Set Label

A data set label can be generated in secondary storage when a data set is created. The source of information for a data set label is the updated JFCB, as indicated by line 6 of Figure 15.

Data set labels facilitate subsequent references to a data set. Any characteristics of a data set defined in a data set label need not be redefined in a DCB macro-instruction or in a DD statement.

COMPLETING THE DATA CONTROL BLOCK, JFCB, AND DATA SET LABEL: Figure 15 illustrates, by lines 1 through 6, the steps that the system takes to fill fields of the data control block, the JFCB, and the data set label. The following paragraphs describe each of these steps in sequence.

Line 1 indicates that the data control block is initially created by the DCB macro-instruction; this normally occurs at assembly time. Prior to issuing the OPEN macro-instruction, the data control block can be moved, copied, or modified by the problem program.

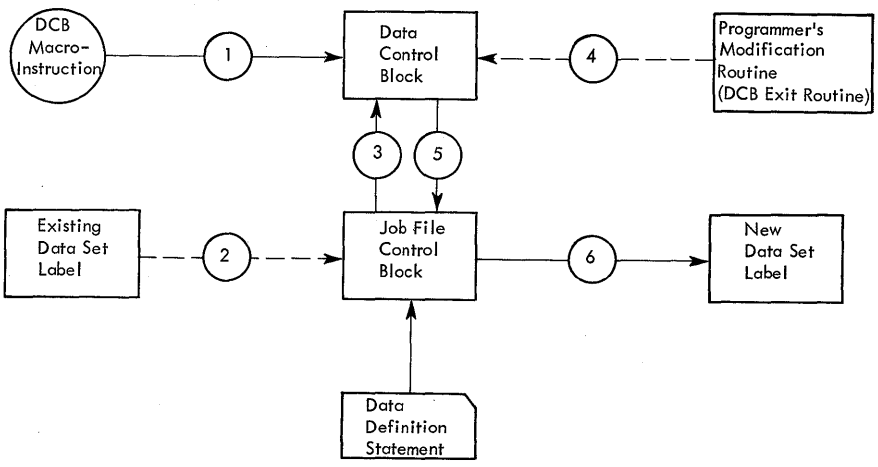


Figure 15. Flow of Information To and From Data Control Block

Line 2 indicates that, for direct-access devices and labeled tape input, an existing data set label is read by the OPEN routines and used to fill empty fields in the JFCB. The data set label contains fields describing data attributes, e.g., block size and record format. (This step does not occur for unit record devices, unlabeled tape, or tape output.)

The JFCB has, in addition to data attribute fields, fields that contain information related to processing, e.g., number of buffers to be used and error options that were chosen. Line 3 indicates that, at this point, the JFCB is used to fill any empty fields in the data control block. Both data attribute fields and processing-related fields can be filled.

Line 4 indicates that the problem program's data control block exit routine, if present, is entered. This routine can either fill any data control block fields that are still empty or modify other fields in the data control block. It then issues a RETURN macro-instruction to return to an OPEN routine.

Line 5 indicates that after the OPEN routines prepare the problem program to access the data set, any empty fields in the JFCB are filled from the data control block. Both data attribute fields and processing-related fields can be filled.

Line 6 indicates that for output on direct-access volumes, the JFCB is used to fill any empty fields in the data set label, and for output on labeled tape volumes, the JFCB is used to construct a new data set label. Only data attribute fields are used.

When the data control block is closed, all information that entered it from either the JFCB or the data set label is removed. The JFCB is not returned to its original condition.

DCB -- Define Data Control Block

The DCB macro-instruction reserves space for a data control block at assembly time and causes specified fields describing data set characteristics and processing requirements to be placed into that data control block. The number of fields that can be specified in this macro-instruction depends upon the organization of the associated data set. The fields that may be specified which are common to all data set organizations (except telecommunications) provide the following types of information:

- Symbolic name of the DD statement that identifies the data set.
- Data set organization.
- Record format.
- Principal macro-instructions to be used for input/output operations on the data set.
- Options to be selected.
- Number and size of buffers.

Particular organizations may require such supplementary specifications as:

- Tape density (sequential organization).
- Number of tracks in cylinder overflow area (indexed sequential organization).
- Address of area containing terminal address (telecommunications organization).
- Number of tracks or blocks to be searched for a block (direct organization).

In addition to information regarding data set characteristics and processing requirements, the programmer can specify certain addresses in the DCB macro-instruction. These addresses identify:

- The location of a routine that performs end-of-data set procedures.
- The location of a routine that supplements the system's error recovery routine.
- The location of an exit list.

The following sections briefly describe the items listed above.

END-OF-DATA SET ROUTINE: For data sets that are read sequentially, the programmer must write an end-of-data set routine that performs any desired final processing.

SYNCHRONOUS ERROR EXIT ROUTINE: This routine supplements the system's error recovery routine. To analyze exceptional conditions and uncorrectable errors, the programmer may write a routine that is entered if an input/output operation ends abnormally.

EXIT LIST: The exit list can contain the addresses of routines that process user

labels, the address of a routine that modifies the data control block, and checkpoint information. Complete details on exit lists are in the publication IBM System/360 Operating System: Control Program Services. The following sections describe the three types of entries in the exit list.

Standard User Label Exit Routines: With sequential organizations, the programmer may write routines for user labels that:

- Verify user header labels (UHL1-8).
- Verify user trailer labels (UTL1-8).
- Create user header labels (UHL1-8).
- Create user trailer labels (UTL1-8).

After verification or creation of any user label, the routine returns control to the system (by means of a RETURN macro-instruction) with an indication of whether another user label is to be read or written.

If the programmer does not wish to verify all existing user labels, automatic repositioning takes place.

Data Control Block Exit Routine: To change fields in the data control block after it has been filled, or to add optional information, the programmer writes a routine (see Figure 15) that is entered when the data control block is opened. Among the fields that may be modified by such a routine are those for buffering technique, block length, and addresses of user exit routines. The programmer also might use a data control block exit routine to determine data set characteristics by examining fields filled by data set labels. The data control block exit routine must return control to OPEN (by means of a RETURN macro-instruction). To assist the programmer in modifying a data control block, the system provides a DCBD macro-instruction, which is described in the publication IBM System/360 Operating System: Control Program Services.

Address of Checkpoint Data Control Block: If an automatic checkpoint is to be taken during end-of-volume processing, the programmer specifies the address of the data control block for a checkpoint data set.

DD Statement/DCB Macro-Instruction Relationship

For ease and flexibility in filling data control block fields, the programmer should use the DCB macro-instruction to specify information that remains constant for all data sets, using a given data control block, that are to be processed by the associated program. Since a different DD statement can be inserted each time the data set is processed, the programmer should place into the DD statement the information about the data set that may vary each time the program is used.

Corresponding to the wide range of system facilities available for control of data is an equally wide range of facilities for access to that data. The variety of techniques for gaining access to a data set is derived from two variables: data set organization and macro-instruction language. Data set organizations have been discussed previously; the macro-instruction languages are discussed in this section.

MACRO-INSTRUCTION LANGUAGES

The programmer requests input/output operations on data sets through macro-instructions that are divided into two categories or languages: the language for basic access and the language for queued access. Each language is identified according to its treatment of anticipatory buffering and input/output synchronization with processing. The combination of a language (and an associated set of facilities for the storage and retrieval of a data set) and a given data set organization is called an access method. In choosing an access method for a data set, therefore, the programmer must consider not only its organization, but also the macro-instruction language capabilities. The in-line code generated by the macro-instructions for both languages is optionally reenterable, depending only upon the form in which parameters are expressed. A discussion of each language and its characteristics follows.

LANGUAGE FOR QUEUED ACCESS

The language for queued access provides the GET and PUT macro-instructions for transmission of data between main and secondary storage. These macro-instructions cause automatic blocking and deblocking of the logical records they store and retrieve. Anticipatory buffering and synchronization of input and output with CPU processing are two distinctive automatic features of the queued language.

System-controlled anticipatory buffering permits the programmer to use as many input or output buffers as he needs without issuing multiple GET or PUT macro-instructions to fill or empty buffers. The system automatically sets up the requested number of buffers, and provides overlap of input/output operations with CPU processing by automatically filling or emptying buffers at the earliest possible times. Therefore, more than one input block normally is in main storage at the same time to prevent input/output operations from delaying record processing.

With automatic synchronization of input and output with processing, the programmer need not test for completion of input/output operations, input/output errors, or exceptional conditions. After a GET or PUT macro-instruction is issued, the system does not return control to the problem program until an input area is filled, or an output area is available. Exits to error correction and end-of-volume or end-of-data set procedures are automatically taken when the corresponding condition occurs.

LANGUAGE FOR BASIC ACCESS

The language for basic access provides the READ and WRITE macro-instructions for transmission of data between main and secondary storage. Although anticipatory buffering and input/output synchronization with processing are not provided automatically by the system, the programmer can perform and control these functions.

Buffers may be allocated by either the system or the programmer. Each buffer is filled or emptied when a READ or WRITE macro-instruction is addressed to it.

Because the macro-instructions that request data transmission only initiate input/output operations, the programmer cannot assume, after issuing a READ or WRITE macro-instruction, that the operations are completed or that appropriate exits are taken. To test for completion of input/output operations and for errors and exceptional conditions, the programmer interrogates, either directly or through a macro-instruction, the fields of a main storage area called a data event control block.

Data Event Control Block (DECB)

A data event control block (DECB) is a main storage area, reserved by either the programmer or the system, that relates an input/output operation to a specific READ or WRITE macro-instruction. Each READ or WRITE macro-instruction requires one data event control block that contains control information and pointers to status indicators. The programmer waits for completion of the input or output operation and tests for successful completion by issuing appropriate macro-instructions. After the input/output operation requested by a READ or WRITE macro-instruction is completed, a subsequent READ or WRITE macro-instruction may use the same data event control block.

Fields of the data event control block may be defined as parameters of the corresponding READ or WRITE macro-instruction and stored by the system, or may be filled in directly by the programmer. Once a parameter is placed into the data event control block, it need not be redefined by subsequent READ or WRITE macro-instructions unless its value is to be changed.

CLASSIFICATION OF ACCESS METHODS

As previously stated, an access method results from the integration of a language and an associated set of facilities for storing and retrieving a data set of a given organization. Access methods can be identified primarily by the data set organizations to which they apply. For example, a combination of sequential organization with either language is termed a sequential access method (SAM). A particular access method, however, requires further qualification as either queued or basic. Thus, the access method that uses the macro-instructions for queued access for operations on sequentially organized data is called the queued sequential access method (QSAM).

Table 1 identifies each available data access method as a combination of a language and a data set organization.

Various access methods may, in many cases, be used with the same data sets. For example, a given data set might be defined to have a partitioned organization for one purpose, and a sequential organization for another, and thus might be retrieved or stored by QSAM, BSAM or BPAM.

Table 1. Data Access Methods

Data Set Organization	Macro-Instruction Language	
	Basic	Queued
Sequential	BSAM	QSAM
Partitioned	BPAM	
Indexed Sequential	BISAM	QISAM
Direct	BDAM	
Telecommunications[1]	BTAM	QTAM

[1]QTAM and BTAM are described in the publication IBM System/360 Operating System: Telecommunications.

EXECUTE CHANNEL PROGRAM (EXCP) ACCESSING PROCEDURE

The system provides for scheduling and queuing of input/output requests, efficient use of channels and devices, data protection, interruption procedures, and error recognition and retry. Through the EXCP (execute channel program) macro-instruction, the programmer can utilize these system functions to control directly an input/output device for access of any data set organization, without using a specific access method or language for access.

When using EXCP, the programmer provides a channel program, which is a list of channel command words appropriate to the desired device and operation. The programmer also provides an input/output control block (IOB) that is used with the channel program to maintain status information regarding completion of the operation. If an input/output error results from an EXCP macro-instruction, use of the system's standard error recovery procedures depends on the complexity of the channel program.

The EXCP macro-instruction gives the programmer more freedom in controlling devices than do the access methods, yet retains many of the advantages of working with the operating system. To use EXCP successfully, however, the programmer needs detailed knowledge of device control, system functions, and control block structure. For example, a programmer using a direct-access device with EXCP must be familiar with the techniques and control blocks required for control of secondary storage; control of magnetic tape with EXCP requires an understanding of how volume switching is initiated.

Any device controlled with EXCP must be supported by corresponding device-dependent programs and input/output error routines in

the system. For devices not supported by the system, the installation must supply appropriate programs to be included in the operating system when it is generated. Further details about EXCP are in the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550.

OPEN AND CLOSE MACRO-INSTRUCTIONS

Before applying an access method to a data set, the programmer must request that a data control block be completed as a logical connection between the data set and the problem program. The programmer issues an OPEN macro-instruction that completes the data control block fields, establishes address relationships and linkages to access routines, issues mounting messages to the operator, verifies or creates data set labels, interrogates tape volume labels, positions volumes to the first record to be processed, allocates buffer pools as required, and begins filling buffers for data sets to be read using the queued sequential access method.

To accomplish these functions, the routine that is executed as a result of the OPEN macro-instruction requires access to information that the programmer supplies in a DD statement. As previously described, this information is stored in a JFCB. The execution of the OPEN macro-instruction causes the system to read the appropriate JFCB from a job queue into main storage for processing by the OPEN routine. The JFCB is then returned to the job queue for subsequent use by other system functions.

After input/output operations on a data set have terminated, the programmer logically disconnects the data set from the problem program by issuing a CLOSE macro-instruction. This macro-instruction closes the data control block, handles volume dispositions, creates data set labels, ensures the writing of queued output buffers, and relinquishes main and secondary storage. After a data control block has been closed, it may be used for another data set.

For a data set that is to be processed by the basic sequential access method, the system provides a variation of the CLOSE macro-instruction, CLOSE (TYPE=T). In executing this macro-instruction for data sets on magnetic tape or direct-access volumes, the system processes labels and repositions volumes as required, but does not logically disconnect the data set from the problem program. As a result, the processing of a data set for which a CLOSE (TYPE=T) macro-instruction has been issued can be continued at a later stage in the problem program without the programmer having to reissue an OPEN macro-instruction. If a data set is to be processed more than once by a problem program, the programmer can improve performance by issuing the CLOSE (TYPE=T) macro-instruction.

For a data set to be processed, the programmer must issue an OPEN macro-instruction to initialize the associated data control block. If the programmer does not close a data control block with a CLOSE macro-instruction, the system automatically closes it when the task terminates if the data control block is available to the system at this time. If it is not, the task is abnormally terminated.

An OPEN or a CLOSE macro-instruction may be addressed to more than one data control block. Opening or closing several data control blocks is faster than issuing a separate OPEN or CLOSE macro-instruction for each data control block, but the latter technique uses less main storage space than the former. The amount of main storage space required for the OPEN and CLOSE routines is directly proportional to the number of data control blocks being processed; a portion of this space is used only when the OPEN or CLOSE routines are in operation and is released before control is returned to the problem program.

Use of OPEN and CLOSE

The programmer specifies, in the OPEN macro-instruction, the intended use of a data set. The functions of the OPEN and CLOSE routines vary according to the use of the data set by the problem program.

Table 2 summarizes, for each applicable access method, every data set use (mode) that may be specified in the OPEN macro-instruction, and the representation of each use as a parameter value. The modes are ignored for the queued indexed sequential and the basic indexed sequential access methods.

The programmer also specifies, as a parameter value, volume positioning for end-of-volume conditions in the OPEN macro-instruction, volume positioning for end-of-data set conditions in the CLOSE macro-instruction, and data set repositioning in the CLOSE (TYPE=T) macro-instruction.

For each access method, positioning is specified as either of the following:

- LEAVE - leave the volume positioned at the logical end of the portion of the data set just read or written on that

volume. (The logical end is either the end or the beginning of this data set portion, depending on whether forward or backward reading is used for an input data set.)

- REREAD - reposition the volume at the logical beginning of the portion of the data set just read or written on that volume.

Either option causes positioning to a point outside of the data (and label) portion of the data set, with one exception: repositioning for CLOSE (TYPE=T) is to a point within the data portion.

Volume disposition specified in the OPEN or CLOSE macro-instructions can be overridden by the system depending upon the availability of devices at a particular time. However, the problem programmer need not be concerned when this situation arises, because the system automatically requests the mounting and dismounting of the appropriate volumes.

Table 2. Data Set Uses Specified in OPEN Macro-Instruction

OPEN Parameter Value[1]	Data Set Use
Queued Sequential Access Method (QSAM)	
INPUT	Data set is read forward sequentially. Labels are processed as input.
OUTPUT	Data set is written sequentially. Labels are processed as output.
RDBACK	Data set on magnetic tape is read backwards sequentially. Labels are processed as input.
UPDAT	Data set on direct-access volume is read sequentially and can be updated-in-place by output requests that write back to the data set the last record read. Labels are processed as input.
Basic Sequential Access Method (BSAM)	
INPUT	Data set is read sequentially either forward or backward (depending on READ macro-instruction operand). Labels are processed as input.
OUTPUT	Data set is written sequentially. Labels are processed as output.
RDBACK	Data set on magnetic tape is read sequentially either forward or backward (depending on READ macro-instruction operand). Labels are processed as input.
UPDAT	Data set on direct-access volume is read sequentially and can be updated-in-place by output requests that write back to the data set the last record read. Labels are processed as input.
INOUT	Data set on magnetic tape or direct-access volume is read sequentially. Labels are normally processed as input. If records have been written to the data set, subsequent labels are processed as output. The volume is repositioned, and the data control block remains open so that the data set can be processed again as output.
OUTIN	Data set is written sequentially on magnetic tape or direct-access volume. Labels are processed as output. The volume is repositioned, and the data control block remains open so that the data set can be processed again as input.
Basic Direct Access Method (BDAM)	
INPUT	Data set on direct-access volume is read.
OUTPUT	Data set is written on direct-access volume. OUTPUT is treated as if UPDAT were specified.
UPDAT	Data set on direct-access volume is read or written. Blocks can be updated-in-place or added to the data set.
Basic Partitioned Access Method (BPAM)	
INPUT	Data set on direct-access volume is read sequentially.
OUTPUT	Members of data set are written sequentially on direct-access volume.

[1]If a parameter value is not specified, INPUT is assumed.

BUFFERS AND BUFFER POOLS

A buffer is a main storage area used for intermediate storage of input/output data. Usually its size corresponds to the size of the blocks in a data set using the buffer. The part of a buffer which contains a logical record is called a buffer segment. A buffer pool is a group of buffers connected so that a data set associated with the pool can be used with any of the available buffers.

The programmer can construct a buffer pool using either of the two techniques discussed in the following text. These techniques are common to all access methods and must be used prior to opening the data control block of the data set (or during the optional data control block exit routine). However, each access method provides additional facilities for automatic buffer pool construction when a data control block is opened if neither technique is used. (These facilities and the procedures for obtaining individual buffers from a pool are discussed in detail under "Buffering Considerations" in the sections of this publication describing the individual access methods.)

ASSEMBLY TIME BUFFER POOL CONSTRUCTION

When the programmer knows at assembly time both the number and size of the buffers he wants to associate with a given data set or data sets, he can reserve an area of appropriate size to be subsequently used as a buffer pool. During execution of a problem program, he can structure this area into a buffer pool by issuing a BUILD macro-instruction. Any type of area, from a storage area defined by the programmer prior to execution time to an area containing coding that is no longer required, can be structured into a buffer pool.

BUILD -- Build a Buffer Pool

The BUILD macro-instruction structures a main storage area reserved by the programmer into a buffer pool.

The address of a main storage area to be used as a buffer pool is specified in the BUILD macro-instruction. This address must also appear in the data control block associated with each data set that will use the pool. The number of buffers to be contained in the pool and the byte length of each buffer must also be specified in the BUILD macro-instruction.

When the data control block of the data set using the pool is closed, the pool area can be reused as required. The programmer, however, will frequently find it useful to restructure this area (by issuing another BUILD macro-instruction) into a new buffer pool to be assigned to another data set.

The area reserved for use as a buffer pool can be assigned to two or more data sets that are able to use buffers of the same length. If this is done, the area should be large enough to accommodate the total number of buffers that will be required at any one time during program execution. For example, if two data sets to be processed simultaneously require five buffers each, the size of the area specified should be ten buffer lengths.

OBJECT TIME BUFFER POOL CONSTRUCTION

When a programmer does not want to reserve a specific area for use as a buffer pool, he can request the system to obtain and structure an appropriate area in main storage by issuing a GETPOOL macro-instruction.

GETPOOL -- Get a Buffer Pool

The GETPOOL macro-instruction structures a main storage area obtained by the system into a buffer pool, and assigns that area to a data control block associated with a specific data set.

The address of the data control block, the number of buffers required in the pool, and the byte length of each buffer in the pool must be specified in the GETPOOL macro-instruction.

In many applications, single- or double-word alignment of a block within a buffer is important. The programmer can specify (in the DCB macro-instruction) that the buffers are to start either on a double-word or on a full-word boundary that is not also a double-word boundary. If double-word alignment is specified for format V logical records, the fifth byte of the first logical record of each block is aligned on a double-word boundary. The alignment of subsequent logical records in each block depends upon the lengths of the preceding records of that block. If double-word alignment is required for the first byte of the first logical record in a block, full-word buffer alignment should be specified.

When the buffer pool is no longer required, the area obtained with the GETPOOL macro-instruction should be returned to the system by issuing a FREEPOOL macro-instruction.

FREEPOOL -- Free a Buffer Pool

The FREEPOOL macro-instruction returns a buffer pool, previously obtained automatically or by a GETPOOL macro-instruction, to main storage.

The address of the data control block associated with the data set that used the pool must be specified in the FREEPOOL macro-instruction.

ACCESS METHODS FOR SEQUENTIAL DATA SETS

Two access methods, the queued sequential and the basic sequential, are provided to enable the programmer to store and retrieve the records of a sequential data set. Both access methods can be used when data is processed on unit record equipment, paper tape readers, magnetic tapes, and direct-access devices. These access methods are particularly valuable in such applications as copying data sets from one volume to another, updating payroll or inventory data sets, and merging two or more sequential data sets.

DATA FORMAT-DEVICE TYPE RELATIONSHIPS

The following text discusses data format considerations that apply to specific input/output device types supported by the queued sequential and basic sequential access methods. Included are descriptions of acceptable data formats for each of the devices supported.

Card Readers and Punches

Readers and punches transfer data of any record format (F, V, or U). When format V records are punched, the initial control bytes in the buffer are ignored, and not punched. The control character C, if specified, is used for stacker selection only; it is not punched.

When the system transfers data from main storage to be punched, each card punched corresponds to one logical record; for this reason, if the programmer plans to have cards punched, he should restrict the maximum size of his records to 80 or 160 data bytes, depending upon whether the mode is EBCDIC or column binary, respectively.

When the queued sequential access method is used, punch error correction on the IBM 2540 Card Read Punch is automatically performed only for data sets with three or more buffers.

Printers

The printer accepts data of any record format (F, V, or U). When format V records are printed, the initial control bytes in the buffer are ignored, and not printed. The control character C, if specified, is used for carriage control only; it is not printed.

Each line of print corresponds to one logical record in storage; hence, if a data set is to be printed, the length of its records should not exceed one line of print.

The system does not automatically perform initial positioning of the printer carriage.

Paper Tape Reader

The paper tape reader reads format U and format F records. Each format U record on paper tape is followed by an end-of-record character. This character is not required for format F records. Data read from paper tape is optionally converted into the System/360 internal representation of any of six standard paper tape codes.

Magnetic Tape

All record formats (F, V, and U) are acceptable to magnetic tape; all control bytes are transmitted. 7-track tapes not using the data conversion feature do not handle format V. (It is recommended that all blocks be at least 16 bytes in length.)

Direct-Access Devices

All record formats (F, V, and U) are acceptable to direct-access devices; all control bytes are transmitted.

All direct-access devices have the same track format. Each track consists of control information, a capacity record (R_0),

and the records (R_1 - R_n). Additional track format information follows:

1. Record R_0. The capacity record is formatted according to a system standard. Only the basic sequential access method can be used to rewrite the data portion of this record.

2. Record R_1. This record is either the first complete data record on a given track, or the overflow portion of a record from a preceding track (if the track overflow option is in effect). It has a record number of 1 in the count field.

3. Other Data Records. The record number in the count field can extend sequentially from 2 to a maximum of 255.

Both the queued and basic sequential access methods can be used to store and retrieve the data records of a direct-access volume. Normally, as many records are stored on a track as is physically possible. A data set can be retrieved in its entirety even if the tracks on which the data set resides do not each contain a full complement of records.

CHAINED SCHEDULING

Both the queued sequential and basic sequential access methods permit the programmer to accelerate the input/output operations required for a data set through the use of a technique known as chained scheduling. This technique, specified in the DCB macro-instruction, bypasses the normal input/output scheduling routines of the system to dynamically chain several input/output operations together. A series of separate read operations, for example, functioning under chained scheduling, is issued to the computing system as one continuous read operation using the PCI flag in command words for synchronization.

Chained scheduling increases input/output performance by reducing both CPU time and channel start and stop time required to transfer records between main and secondary storage. It also can sharply reduce the effects of rotational delay since several successive blocks, independently called for, can be retrieved in a single rotation. The use of chained scheduling, however, restricts the programmer in his choice of buffering techniques. In addition, each data set for which chained scheduling is specified must be assigned at least two, and preferably three, buffers.

Chained scheduling is most valuable for programs that are input/output limited. If the programmer specifies chained scheduling for a data set, the input/output operations of the data set may intermittently monopolize available time on a channel. For this reason, if more than one data set is to be processed, the programmer should assign each data set to a separate channel, if possible.

QUEUED SEQUENTIAL ACCESS METHOD

The queued sequential access method permits the programmer to store and retrieve the records of a sequential data set without coding blocking/deblocking and buffering routines. The programmer can, therefore, concentrate all his efforts on processing the data he reads and writes. Another major feature of this access method is that it provides two buffering techniques, allowing the programmer to choose the one most suited to his application.

The macro-instructions of the queued sequential access method are, for the most part, device-independent, permitting the programmer to write programs that can be executed using a number of different input/output devices.

RECORD FORMATS

The queued sequential access method supports five block configurations:

1. Format U.
2. Format F, Unblocked.
3. Format F, Blocked.
4. Format V, Unblocked.
5. Format V, Blocked.

BUFFERING CONSIDERATIONS

The following section discusses the buffering facilities available with the queued sequential access method. Descriptions of buffer pool construction and buffer assignment procedures are included.

Buffer Pool Construction

The system automatically assigns a buffer pool to a data control block when it is opened, if the programmer has not assigned such a pool prior to the conclusion of the optional data control block exit routine. The programmer must return this pool area

to main storage when it is no longer required by issuing a FREEPOOL macro-instruction (refer to the section "Buffers and Buffer Pools").

When the programmer requires buffers for specialized applications, he should use either the GETPOOL or the BUILD macro-instruction (refer to the section "Buffers and Buffer Pools").

Buffer Assignment Procedures

After the buffer pool has been constructed, the programmer requests input or output of data records. The system automatically blocks and deblocks data records, obtains and controls buffers, and reads and writes data records, as required. If a buffer pool constructed by the BUILD macro-instruction is used by more than one data set, individual buffers from the pool are automatically assigned to the appropriate data control block according to the programmer's specifications.

Buffers are returned to the pool area by the system when the programmer closes the data control block of the data set using the pool.

BUFFERING TECHNIQUES

The term buffering technique refers to the manner in which buffers are assigned to, and used with, input and output data sets. Two buffering techniques are provided: simple and exchange. Each is described in detail in the following text. Correct choice of these techniques contributes to the efficiency of the problem program.

Simple Buffering

In simple buffering, a data set is associated with a specific group of buffers. A data set always uses buffers obtained from the pool assigned to its data control block at the time it is opened. Logical records can be treated in any of the following ways:

- Moved between a buffer and an independent work area.
- Processed within a buffer.
- Moved from an input buffer to an output buffer.

The programmer, therefore, can insert new records or delete old records, as required, when creating a new data set from an existing data set.

Simple buffering is the most widely applicable of the two techniques provided. The system places no restrictions on the record formats or macro-instructions that can be used with this technique.

REQUESTING SIMPLE BUFFERING: By any of the methods described previously, the buffer pools are created and associated with data control blocks. The programmer must indicate in the data control blocks of the data sets that simple buffering is to be used.

Exchange Buffering

Exchange buffering combines the record insertion/deletion features of simple buffering with certain device features associated with data chaining ("scatter read" and "gather write"). The system implements "scatter read" by issuing separate chained channel command words for each logical record in a block, reading these records into separate main storage locations (buffer segments or work areas). "Gather write" is implemented similarly. The system issues separate chained channel command words to write logical records from separate main storage locations into one block on a volume.

With exchange buffering, logical records can be handled in any of the following ways:

- Processed within an input buffer segment.
- Moved from an input buffer segment to a work area.
- Moved from an input buffer segment to an output buffer segment.
- Moved from a work area to an output buffer segment.
- Included in a scatter read.
- Included in a gather write.

Exchange buffering can be requested only for programs that are to process either unblocked records or blocked format F records. When blocked records are processed, the implementation of exchange buffering is totally dependent upon data chaining. The ability to chain data is related to a combination of such factors as the speed of the central processing unit, the input/output device selected, and the

type of input/output channel involved. If the programmer has requested exchange buffering, and data chaining cannot be effected, simple buffering is substituted automatically. If simple buffering is substituted, alignment is guaranteed only if logical records are multiples of double words.

For a data set using exchange buffering and blocked format F records, the system segments the buffers so that each segment is of the same length and has the same word boundary alignment. (The programmer must ensure that buffers are large enough for boundary alignment of segments.) When the programmer creates a buffer pool whose individual buffers are each larger than the blocks of the data set, each buffer segment will be larger than the record it contains. By constructing such a buffer pool, the programmer can increase the length of each record within its buffer segment as the need arises. This technique is especially useful when the GET and PUT macro-instructions are used in substitute mode.

Exchange buffering is useful and efficient in applications that involve either merging two or more data sets, or updating one data set by inserting or deleting logical records.

REQUESTING EXCHANGE BUFFERING: Using any of the methods described previously, the programmer creates and associates a buffer pool with both the input and output data sets.

Caution: Exchange buffering cannot be requested for data of blocked format F records when the programmer has specified the track overflow option.

MACRO-INSTRUCTIONS

The following macro-instructions are provided by the queued sequential access method for input/output operations.

GET -- Get a Logical Record

The GET macro-instruction obtains a logical record from an input data set in either of three modes of operation: move, locate, or substitute. In the move mode, GET moves the logical record from an input buffer into a work area specified by the programmer. The record may be processed or extended in the work area. Move mode GET can be used only with simple buffering.

In the locate mode, GET does not move the logical record from the input buffer, but places into a standard register the address of the buffer segment in which the programmer may process that record. The programmer may not extend record size.

In the substitute mode, GET interchanges the addresses of the programmer's work area and the buffer segment containing the logical record, so that the buffer segment's address is placed in a standard register and the work area becomes a segment of the input buffer. Substitute mode GET can be specified only if exchange buffering is specified. If exchange buffering cannot be performed because data chaining cannot be effected, substitute mode GET works identically to move mode GET with simple buffering.

The programmer may specify only one of these modes of operation as a parameter of the DCB macro-instruction associated with the input data set. Table 3 indicates the valid combinations of modes of GET with each buffering technique.

GET operates in a strictly sequential and device-independent manner. As required, GET schedules the filling of input buffers, deblocks records, and directs input error recovery procedures. After GET has retrieved all records to be processed and has discovered that no data remains, the system automatically checks labels and passes control to the programmer's end-of-data set exit specified in the data control block. GET also tests for an end-of-volume condition and initiates automatic volume switching if an input data set or a concatenation of input data sets extends across several volumes. Finally, GET automatically resolves data set discontinuities within the same volume.

The following operands must be specified by the programmer in the GET macro-instruction:

1. The address of the data control block associated with the input data set.

2. The address of the programmer's work area for input logical records if the mode of operation is move or substitute.

Note 1: Optionally, GET may read backwards from magnetic tape.

Note 2: Only move mode GET is supported for the paper tape reader when conversion is specified.

RELSE -- Release an Input Buffer

The RELSE macro-instruction causes the GET macro-instruction to ignore the remaining logical records in an input buffer and to obtain logical records from the next buffer. When the programmer does not require the remaining contents of an input buffer that GET is deblocking, he may issue the RELSE macro-instruction to release the buffer so that the next logical record is retrieved from another block. When used in conjunction with move mode GET, RELSE causes the input buffer to be scheduled for refilling immediately. When used with locate mode GET, RELSE prevents refilling of the buffer until a subsequent GET is issued. RELSE does not affect buffers containing unblocked records.

The address of the data control block associated with the input data set must be specified in the RELSE macro-instruction.

PUT -- Put a Logical Record

The PUT macro-instruction places a logical record into an output data set in either of three modes of operation: move, locate, or substitute. In the move mode, PUT moves the logical record from a work area specified by the programmer into an output buffer. In the locate mode, PUT does not move the logical record into the output buffer, but places into a standard register the address of the buffer segment into which the programmer may move that record. Locate mode PUT can be used only with simple buffering (Table 3).

In the substitute mode, PUT interchanges the addresses of the work area containing the logical record and a free buffer segment. The buffer segment's address is placed into a standard register and the work area effectively becomes a segment of the output buffer. Substitute mode PUT can be specified only if exchange buffering is specified. If exchange buffering cannot be performed because data chaining cannot be effected, substitute mode PUT works identically to move mode PUT with simple buffering.

The programmer specifies only one of these modes of operation as a parameter of the DCB macro-instruction associated with the output data set. Table 3 indicates the valid combinations of modes of PUT with each buffering technique.

Like the GET macro-instruction, PUT operates in a strictly sequential and device-independent manner. As required, PUT blocks records, schedules the emptying of output buffers, and handles output error correction procedures. PUT also resolves discontinuities of available space, tests for an end-of-volume condition, and initiates automatic volume switching and label creation.

The following operands must be specified in the PUT macro-instruction:

1. The address of the data control block of the output data set.

2. The address of the programmer's work area for output logical records if the mode of operation is move or substitute.

Note: If PUT is directed to a card punch or printer, the system automatically adjusts the blocking factor of format F or V blocks to 1. This allows the programmer to specify a record length and a buffer length that provide an optimum blocking factor for possible intermediate tape or direct-access devices.

PUTX -- Put From Existing Data Set

The PUTX macro-instruction is used to update an input data set in place or to create an output data set based on an input data set. PUTX updates, replaces, or inserts records read from existing data sets but does not add or create records from other sources. According to one of two uses, update or output, and depending on the buffering technique, PUTX causes a

Table 3. Valid Combinations of Modes and Buffering Techniques

Simple Buffering Only	Both Simple and Exchange Buffering	Exchange Buffering Only
GET move PUT locate PUTX update	GET locate PUT move PUTX output	GET substitute PUT substitute
Note: Exchange buffering does not permit format V blocked records; format V records cannot be used at all with substitute mode.		

block or a logical record to be either written directly from the input buffer into which it was read or transferred from an input buffer to an output buffer. PUTX must be preceded by a GET macro-instruction in the locate mode that refers to the same input data control block as the PUTX. Table 3 indicates the valid combinations of uses of PUTX with each buffering technique.

When the use is update, PUTX places records back into an input data set. Each PUTX flags the entire input buffer containing the segment addressed by the preceding GET macro-instruction to be written back to the same location in secondary storage from which it was read. The block is written when the first GET is given for the next buffer. PUTX for update use applies only to data sets on direct-access volumes and may be used only with simple buffering.

When the use is output, PUTX places records into an output data set and is applicable to any buffering technique. With simple buffering, PUTX moves a logical record from an input buffer segment to an available output buffer segment. New records can be added to the output buffer by PUT macro-instructions. PUTX, like PUT, schedules the buffer to be written when it is filled.

With exchange buffering, PUTX utilizes data chaining to schedule a gather write of logical records from areas some of which were previously input buffer segments. Correspondingly, buffer segments that were previously used by an output data set are scheduled for a scatter read.

The addresses of the data control blocks of the output and the input data sets (output use), or associated with the data set updated-in-place (update use), must be specified in the PUTX macro-instruction.

TRUNC -- Truncate an Output Buffer

The TRUNC macro-instruction causes the PUT macro-instruction to regard an output buffer as full, and subsequently to place logical records into the next buffer. When the programmer does not need the remaining portion of an output buffer that PUT is blocking, he may issue the TRUNC macro-instruction to truncate the buffer so that the next logical record is placed into another block. Thus, just as input buffers may be released, output buffers may be truncated for writing short blocks.

The address of the data control block of the output data set must be specified in the TRUNC macro-instruction.

The CLOSE macro-instruction effectively truncates the last block of a data set.

As noted in the section "Fixed-Length (Format F)," data sets with truncated blocks are not read from direct-access devices as efficiently as standard F data sets.

Note: When the TRUNC macro-instruction is used with locate mode PUT, the system assumes that a record was placed in the buffer segment pointed to by the previous PUT.

FEOV -- Force End of Volume

The FEOV macro-instruction causes the system to assume an end-of-volume condition for either an input or an output data set, thereby causing immediate automatic volume switching. When volumes are switched, FEOV creates output labels as required and verifies labels on new input volumes.

The address of the data control block of the input or output data set must be specified in the FEOV macro-instruction.

Note: When the FEOV macro-instruction is used with locate mode PUT, the system assumes that a record was placed in the buffer segment pointed to by the previous PUT.

CNTRL -- Control a Printer or Stacker

The CNTRL macro-instruction provides either of two device-dependent control functions: on-line card reader stacker selection or on-line printer carriage control. If directed to a card reader, CNTRL must follow every GET macro-instruction directed to that card reader for the same data set.

The following operands must be specified in the CNTRL macro-instruction:

1. The address of the data control block of the input or output data set.
2. The action to be taken (either stacker selection, line spacing, or channel skipping).
3. A number indicating either the stacker, amount of lines, or the printer carriage tape channel.

Note: If CNTRL is directed to a card reader, only one input buffer may be used, and the preceding GET macro-instruction must be in move mode. As soon as CNTRL is

issued, the buffer is scheduled for refilling.

PRTOV -- Test for Printer Overflow

The PRTOV macro-instruction tests overflow indicators for on-line printer channel overflow. If an overflow indicator is on, PRTOV causes either an automatic skip to a new page or a transfer of control to an optionally specified logical point in the problem program.

The following operands must be specified in the PRTOV macro-instruction:

1. The address of the data control block of the output data set.
2. The printer channel to be tested for overflow (either 9 or 12).
3. The address of a routine may optionally be specified for transfer of control on condition of overflow; if this is not specified, a printer overflow condition causes automatic skipping to channel one.

ERROR CONDITIONS

If data transmission to or from an input/output device is not successful the first time it is attempted, standard error recovery routines, provided by the operating system, attempt to clear the failure and allow the program to continue uninterrupted. An uncorrectable error usually terminates the problem program, but the programmer can specify, in the DCB macro-instruction, other actions to be taken in case of an uncorrectable error. These actions differ for input and output operations, and are described, by type, in the following text. The programmer cannot attempt to reread or rewrite the record in error, since automatic retry is handled by the system.

Input Operations

For uncorrectable errors that occur in filling an input buffer, the programmer can direct the system to:

- Deblock the buffer as though its records had been read correctly.
- Ignore the buffer, and read and deblock the next block.
- Terminate the task.

Output Operations

For uncorrectable errors that occur in transferring data from an output buffer to secondary storage, the programmer can direct the system to:

- Accept the record in error (applies only to printer).
- Terminate the task.

ERROR CONDITIONS: INTERNAL DETAILS

The following more detailed discussion is provided for programmers who code their own error analysis routines. The address of this routine must be specified in the DCB macro-instruction.

Input Operations

When an uncorrectable error occurs after a GET macro-instruction has been issued to transfer data to an input buffer, the system ceases to schedule additional buffers to be filled. It continues, however, to deblock the records of those buffers already filled. When the buffer in which the error occurred is to be deblocked, control is passed to the programmer's error analysis routine.

Output Operations

When an uncorrectable error occurs in transferring data from an output buffer, the system ceases to schedule additional buffers for writing. It continues, however, to block the programmer's data until all output buffers except one are filled. When the system attempts to fill the buffer in which the error occurred, control is passed to the programmer's error analysis routine.

Error Analysis Routine (Synchronous Error Exit)

Error codes, whose addresses are placed by the system into standard registers, can be examined by the programmer to determine the type of error that has occurred, and the location of the buffer containing the record in error. After the programmer has completed his error analysis and processing, he can return control to the system,

whereupon the standard error option the programmer has specified in the DCB macro-instruction is executed.

As an alternative, the data set may be closed in the error analysis routine, in which case the programmer must not return control to the system.

PROGRAMMING NOTES

The following section describes additional programming factors, not specifically related to any one macro-instruction, that the programmer should consider before coding a program.

Direct-Access Volume Options

Both the track overflow and the write validity check options, described previously, are available to the programmer.

The keys in the records on direct-access volumes cannot be processed using the queued sequential access method.

Update-in-Place

The following rules apply to a data set to be updated-in-place:

1. Only simple buffering can be used.
2. Chained scheduling is not available.
3. Macro-instruction usage is limited to the GET (locate), RELSE, and PUTX (update) macro-instructions.
4. When a PUTX (update) macro-instruction is issued for an individual record within a block, the complete block (as it exists in secondary storage) is scheduled to be overwritten. The block is overwritten after the programmer has completed the input processing of all records in the block.
5. Neither alteration of block length, insertion of new blocks, nor deletion of existing blocks is possible.

Concatenated Data Sets

Two or more sequentially organized data sets can be concatenated, i.e., automatically retrieved by the system and processed successively as a single data set. The following considerations apply to concatenation of sequential data sets:

1. Concatenation applies only to data sets opened for INPUT.
2. A maximum of 255 data sets can be concatenated.
3. Only one data control block is associated with all the data sets concatenated. The programmer enters DD statements for these data sets in the order in which they are to be retrieved. If data sets with unlike characteristics (e.g., block length, record format, etc.), are to be concatenated, the data control block must contain an indication of this.
4. When the data sets to be concatenated have unlike characteristics, the programmer must reissue any input requests that have not been completed at the time the system determines that the data set currently being processed does not contain any more blocks. Such a data set is automatically closed, and the next data set to be processed is opened. The programmer should supply an entry in the exit list for his data control block exit routine, so that control may be passed to it during the opening process. From this routine, the programmer must be able to determine the status of all outstanding input requests; any requests not yet completed must be reissued after the opening process is completed, i.e., they cannot be reissued in the programmer's data control block exit routine. When using the queued sequential access method, the programmer must reissue the last GET issued before the new data set was opened. When using the basic sequential access method, each CHECK macro-instruction should be immediately followed by a test of a program switch whose value indicates whether or not to reissue the associated READ macro-instruction and all other READ macro-instructions that were subsequently issued. The program switch should be set to the "on" status in the data control block exit routine and set "off" whenever the test determines an "on" condition.
5. When the data sets to be concatenated have identical characteristics, i.e., both data and device characteristics, the system performs the concatenating function without performing the closing and opening processes described in item 4. Therefore, the data control block exit is not taken and

the programmer is not aware of when the concatenation process takes place.

6. The programmer's end-of-data set routine is not entered until the last data set has been retrieved.

7. Volume switching is automatically performed as for a single data set on multiple volumes.

8. When two data sets on the same volume are to be consecutively concatenated, the disposition option of the OPEN macro-instruction should be LEAVE or REREAD.

9. User label exit routines are executed for each data set, if requested.

For detailed concatenation procedures, see IBM System/360 Operating System: Job Control Language.

Read Backwards

Data sets on magnetic tape that can be read backwards are supported by the System/360 Operating System. The following considerations apply to such data sets:

1. Simple buffering must be used.

2. Format V records cannot be read backwards.

3. Word boundary alignment of the first data byte of each block is ensured only if the length of each block of the data set is a multiple of a single or double word.

4. In reading backwards, logical records are presented in the same manner as in forward reading, but in reverse sequence.

Blocking of Variable-Length Records

When a PUT (locate mode) is used to block variable-length records, the system maintains a maximum logical record length in the data control block. When the space remaining in an output buffer is less than this maximum logical record length, the records in the buffer are transferred as a block to secondary storage and the system begins filling another buffer. To ensure that each block contains the maximum number of records it can accommodate, the programmer can replace the maximum logical record length maintained in the data control block with the actual length of each logical record about to be transferred to an output buffer.

BASIC SEQUENTIAL ACCESS METHOD

The basic sequential access method provides the programmer with an efficient and flexible means for storing and retrieving the blocks of a sequentially organized data set. The macro-instructions of this access method can be used in a device-dependent manner, supplying the programmer with a set of facilities similar to the true functional nature of the input/output devices supported. Certain macro-instructions (NOTE, POINT, and CONTROL) permit him to gain access to data in other than a strictly sequential order. By somewhat limiting the use of these macro-instructions, however, the programmer can code programs that are completely device-independent. Whether using these macro-instructions in a device-dependent or device-independent manner, the programmer does not require a detailed knowledge of the input/output devices themselves.

A major feature of the basic sequential access method is that it permits the programmer to transfer data from an input/output device directly to a specific area of main storage or, conversely, to transfer data from a specific area in main storage to an input/output device, without first moving it to or from a buffer. This feature is particularly useful if, owing to the large size of the records to be processed, no main storage space is available for buffers.

Because of the additional modes of the OPEN macro-instruction provided for the basic sequential access method, it is effective in applications where records are to be alternately read and written, in rapid succession, from and to a data set used as a temporary extension of main storage.

The basic sequential access method can also be used to store and retrieve blocks directly, in any sequence, from a direct-access or a magnetic tape device. This capability, however, is less flexible than the basic indexed sequential or the basic direct access methods, which are more specialized for direct-access devices.

RECORD FORMATS

The basic sequential access method supports five block configurations:

1. Format U.
2. Format F, Unblocked.
3. Format F, Blocked.
4. Format V, Unblocked.
5. Format V, Blocked.

All blocks are treated as the object of an input or an output request. Although the basic sequential access method does not provide either automatic buffering of blocks or blocking/deblocking of records, the resources of the system permit the programmer to code these routines, if required, with a minimum of effort. Truncated format F blocks can be detected by the system and an indication of such blocks is provided for the programmer's error analysis routine, without using the system's error recovery procedures.

For format V blocks in the basic sequential access method, the first eight bytes of a block (four containing block control information and four containing record control information) must be provided by the programmer.

BUFFERING CONSIDERATIONS

The term buffering is used in a different context in the basic sequential access method than it is in the queued sequential access method. In the basic sequential access method, a buffer is a work area rather than an intermediate storage area. The programmer may use the buffering macro-instructions provided by the system to create and assign work areas (buffers) to the problem program, rather than to reserve and identify specific work areas within the program.

Buffer Pool Construction

If the programmer has not assigned a buffer pool to a data set prior to the conclusion of the optional data control block exit routine, and a buffer pool is required, the system assigns a pool automatically when the data control block is opened. The programmer must return the pool to main storage when it is no longer required by issuing a FREEPOOL macro-instruction (refer to section "Buffers and Buffer Pools").

When the programmer requires buffers for specialized applications, he should use either the GETPOOL or the BUILD macro-instruction (refer to the section "Buffers and Buffer Pools").

Buffer Assignment Procedures

To obtain a buffer from a pool constructed by means of any of the above three techniques, the programmer issues a GETBUF macro-instruction.

When the buffer is no longer required, the programmer must return the buffer to its pool by issuing a FREEBUF macro-instruction.

GETBUF -- Get a Buffer From a Pool

The GETBUF macro-instruction causes a buffer to be obtained from a pool associated with a specific data set.

The address of the data control block of the data set requiring the buffer, and a register into which the system is to place the address of the buffer obtained must be specified in the GETBUF macro-instruction.

FREEBUF -- Return a Buffer to a Pool

The FREEBUF macro-instruction causes a buffer to be returned to the pool from which it was obtained.

The address of the data control block of the data set using the buffer pool, and the register containing the address of the buffer to be returned to the pool must be specified in the FREEBUF macro-instruction.

The programmer need not return buffers to a buffer pool in the order in which they were obtained.

MACRO-INSTRUCTIONS

The following macro-instructions are provided by the basic sequential access method for input/output operations.

READ -- Read a Block

The READ macro-instruction requests that a block be transmitted from an input data set to a main storage area specified by the programmer. As previously explained, the fields in the data event control block are specified as parameters of READ (but may be subsequently replaced by parameters of other READ macro-instructions addressed to the same data event control block). READ operates in a strictly sequential and device-independent manner. To allow over-

lap of the input operation with processing, READ does not wait for completion of the input operation, but returns control to the problem program. If the input data set is defined with key fields, READ transmits the key, immediately followed by the block, into the area. READ also automatically resolves extent discontinuities.

The following two operands must be specified in the READ macro-instruction:

1. The address of the data event control block referred to by the READ macro-instruction.

2. The type of input operation (either forward or backward reading).

The following operands are required in READ macro-instructions that initially place the corresponding fields into a data event control block. In subsequent READ macro-instructions using the same data event control block, they may optionally be specified to override the previously specified values. If an operand is omitted, the system assumes its value is already in the data event control block.

3. The address of a data control block of an input data set.

4. The address of an input area.

5. For format U records, the number of bytes to be transmitted.

WRITE -- Write a Block

The WRITE macro-instruction requests that a block be transmitted from a main storage area specified by the programmer to an output data set. Normally, the fields in the data event control block are initially parameters of WRITE (but may be subsequently replaced by parameters of other WRITE macro-instructions addressed to the same data event control block). Like the READ macro-instruction, WRITE operates in a strictly sequential and device-independent manner. To allow overlap of the output operation with processing, WRITE does not wait for completion of the output operation, but returns control to the problem program. If the key length of an output data set is specified in the data control block, WRITE transmits the key, immediately followed by the data, from the area. WRITE also automatically resolves extent discontinuities.

The following two operands must be specified in the WRITE macro-instruction:

1. The address of the data event control block of the WRITE macro-instruction.

2. The type of output operation.

The following operands are required in WRITE macro-instructions that initially place the corresponding fields into a data event control block. In subsequent WRITE macro-instructions using the same data event control block, they may optionally be specified to override the previously specified values. If an operand is omitted, the system assumes its value is already in the data event control block.

3. The address of a data control block of an output data set.

4. The address of an output area.

5. For format U records, the number of bytes to be transmitted. This overriding length should not exceed the length specified in the data control block.

CHECK -- Wait for and Test Completion of Read or Write Operation

The CHECK macro-instruction must be used to wait for the completion of an input/output operation requested by a READ or WRITE macro-instruction. It is also used to test the data event control block for errors and exceptional conditions. As required, CHECK passes control to the appropriate exits that are specified by the programmer in the data control block for error analysis and end-of-data set, and initiates automatic end-of-volume procedures. The programmer must issue a CHECK macro-instruction to test the input/output operation associated with the data event control block before modifying or reusing it. Similarly, a problem program must check an input/output operation for completion before altering the input or output area in main storage.

The address of the data event control block of a preceding READ or WRITE macro-instruction must be specified in the CHECK macro-instruction.

Note: A WAIT macro-instruction can be issued prior to the CHECK macro-instruction. The WAIT macro-instruction synchronizes input/output operations with processing but does not check the data event control block for errors or exceptional conditions or initiate volume switching. Such a WAIT macro-instruction may be issued for multiple event control blocks.

Successive CHECK macro-instructions that test input/output operations on the same data set should be issued in the same order as the associated READ or WRITE macro-instruction for the data set. (This is not true of WAIT.)

FEOV -- Force End of Volume

The FEOV macro-instruction causes the system to assume an end-of-volume condition for either an input or an output data set, thereby causing immediate automatic volume switching. When volumes are switched, FEOV creates output labels as required, and verifies labels on new input volumes.

The address of the data control block of the data set must be specified in the FEOV macro-instruction.

Note: Before issuing an FEOV macro-instruction for an output data set, the programmer should test all write operations for completion.

CNTRL -- Control On-Line Input/Output Devices

The CNTRL macro-instruction provides one of three device-dependent on-line control functions: card reader stacker selection, printer carriage control, or magnetic tape repositioning.

The following operands must be specified in the CNTRL macro-instruction:

1. The address of the data control block of the data set.
2. The action to be taken (either stacker selection, line spacing, channel skipping, tape block forward spacing, tape block backspacing, forward spacing over a tape mark and back to the end of the preceding record, or backspacing over a tape mark and forward to the following record).
3. A number indicating the stacker, amount of lines, printer channel, or count of records.

PRTOV -- Test for Printer Overflow

The PRTOV macro-instruction tests overflow indicators for on-line printer channel overflow. If an overflow indicator is on, PRTOV causes either an automatic skip to a new page or a transfer of control to an optionally specified logical point in the problem program.

The following operands must be specified in the PRTOV macro-instruction:

1. The address of the data control block of the output data set.
2. The printer carriage tape channel to be tested for overflow (either 9 or 12).
3. The address of a routine may optionally be specified for transfer of control on condition of overflow; if this is not specified, a printer overflow condition causes automatic skipping to channel one.

NOTE -- Provide Position Feedback

The NOTE macro-instruction places into a standard register the position on a volume of the last block read from or written into a data set. This feedback identifies the block for subsequent repositioning of that volume.

The identification that NOTE provides is a block count for magnetic tape; for direct-access volumes, it is the track number relative to the beginning of the data set portion on the volume, and the record number within the track.

The address of the data control block of the data set must be specified in the NOTE macro-instruction.

Notes: The following items should be considered when using the NOTE macro-instruction:

1. Before issuing a NOTE macro-instruction, the programmer should test the last input/output operation for completion.
2. NOTE is normally used to provide information for a subsequent POINT macro-instruction.

POINT -- Point to Block

The POINT macro-instruction causes repositioning of a magnetic tape or direct-

access volume to a specified block within a data set on that volume. A subsequent READ macro-instruction starts to read sequential input with the specified block. Thus, POINT permits reading of the data set from any specified position. A subsequent WRITE macro-instruction starts to write sequential output at the block that has been pointed to.

The following operands must be specified in the POINT macro-instruction:

1. The address of the data control block of the data set.
2. The address of a main storage area containing an identification of a block within the data set (block count for magnetic tape; relative track and record number within the track for direct-access volumes).

BSP -- Backspace One Block

The BSP macro-instruction causes the repositioning of a magnetic tape or direct-access volume backwards one data block on the current volume, thus permitting the rereading or rewriting of a block.

The address of the data control block of the data set must be specified in the BSP macro-instruction.

Notes: The following items should be considered when using the BSP macro-instruction:

1. For direct-access volumes, when a WRITE macro-instruction follows a BSP, the remainder of the contents of the track on which the block is written is destroyed. For magnetic tape, when a WRITE macro-instruction follows a BSP, any information immediately following the block may be destroyed.
2. BSP cannot be used for a data set on a direct-access volume if the track overflow option was used in writing the data set.
3. It is recommended that BSP be used only when it is not possible or not practical to use other macro-instructions for repositioning purposes.
4. All READ and WRITE operations must be checked for completion before BSP is issued.

ERROR CONDITIONS

If the programmer's error analysis routine has been entered as a result of an uncorrectable error in data transmission, the programmer cannot attempt to retry the operation in error since automatic retry is handled by the system.

The programmer can examine error code information (supplied in standard registers) and the block in error to determine the type of error that has occurred. After error analysis and processing is complete, the programmer can return control to the system, whereupon processing resumes with the next block of data. Alternatively, the data set in which the error occurred may be closed, in which case return to the system is not permitted. In general, this alternative is recommended for tape and direct-access output.

PROGRAMMING NOTES

The following section describes additional programming factors, not specifically related to any one macro-instruction, that the programmer should take into account before coding a program.

Direct-Access Volume Options

Both the track overflow and write validity check options, described previously, are available to the programmer.

Update-in-Place

A data set on a direct-access volume may be updated-in-place; i.e., records may be read from the data set, processed, and written back to the same positions from which they were read, without destroying the remaining records on a track. The following rules apply to a data set to be updated-in-place:

- Chained scheduling is not available.
- Macro-instruction usage is limited to the READ, WRITE, CHECK, NOTE, and POINT macro-instructions.
- READ and POINT macro-instructions may be issued in any order, as described previously. Because a WRITE macro-

instruction must be preceded by a READ macro-instruction, new records cannot be added to a data set.

- Neither alteration of record length, insertion of new records, nor deletion of old records is permitted.

Read Backwards and Concatenated Data Sets

Considerations that apply to reading a data set backwards from magnetic tape and to concatenation of sequential data sets are listed in the section "Programming Notes" under "Queued Sequential Access Method."

Device Considerations

Table 4 lists the macro-instructions of the basic sequential access method that can be used for specific input/output devices. The permissible modes of the OPEN macro-instruction are listed in parentheses for each device. An X indicates a valid operation; an E indicates that for a particular row, only one of the columns marked with an E may be selected; an asterisk (*) indicates that CNTRL (tape), NOTE, POINT, and BSP are permitted for data in the system input stream (SYSIN) and in the SYSOUT data set but are ignored; and a double asterisk (**) indicates that PRTOV may be specified but is ignored.

Device Independence

Table 4 can be used as a guide in coding device-independent programs. To achieve device independence, the programmer should select those macro-instructions and OPEN modes that are compatible for two or more devices.

For example, to code a program that is device-independent across magnetic tape and direct-access devices, the programmer can use the READ, WRITE, CHECK, and either the BSP or the NOTE and POINT macro-instructions with the INPUT, OUTPUT, INOUT, or OUTIN modes of the OPEN macro-instruction. Once coded, the program can be executed using either magnetic tape or a direct-access device. Similarly, by selection of appropriate macro-instructions and OPEN modes, device compatibility among reader, printer, punch, tape, and direct-access devices may be achieved.

BASIC PARTITIONED ACCESS METHOD

The basic partitioned access method provides for the storage and retrieval of data associated with a partitioned data set. It allows the programmer to select named members and to eventually read or write the records within those members. Except that the basic partitioned access method is restricted to direct-access devices, its primary characteristics are identical to those of the basic sequential access method. These characteristics include buffer pool construction, buffer assignment procedures, and macro-instructions for direct-access devices.

Additional macro-instructions are provided with this access method to permit the programmer to use the directory of the partitioned data set in locating a member for processing, to change the contents of the directory, and to add, replace, delete, or rename members.

RECORD FORMATS AND BUFFERING CONSIDERATIONS

The record formats for a partitioned data set follow the same rules as either the queued or the basic sequential access method, with the exception that standard F format records may not be used. For further information on the compatibility between the basic partitioned access method and the queued or basic sequential access methods, refer to the section "Basic Partitioned Access Method Compatibility."

Refer to the section "Buffering Considerations" for the basic sequential access method for the discussion of buffer pool construction and buffer assignment procedures.

MACRO-INSTRUCTIONS

The following macro-instructions for operations on a partitioned data set directory are provided by the basic partitioned access method.

FIND -- Position to Member of Partitioned Data Set

The FIND macro-instruction causes the address of the first block of a specified partitioned data set member to be determined so that a subsequent READ macro-instruction causes sequential input of blocks to begin with the member specified.

Table 4. Basic Sequential Access Method Device Considerations

Device and Mode of OPEN	Macro-Instructions					
	READ and CHECK	WRITE and CHECK	CNTRL	PRTOV	NOTE and POINT	BSP
Paper Tape Reader (INPUT)	X					
1442 Card Read Punch[1] (INPUT)	X		X		*	
2501, 2520, and 2540 Card Readers (INPUT or INOUT)	X		X		*	
1442 Card Read Punch[2], 2520 and 2540 Card Punches (OUTPUT or OUTIN)		X				
1442 Card Read Punch[3], (INOUT)	X	X				
Printer (OUTPUT or OUTIN)		X	X	X		
Magnetic Tape (INPUT or RDBACK)	X		E*		E*	E*
Magnetic Tape (OUTPUT)		X	E*	**	E*	E*
Magnetic Tape (INOUT or OUTIN)	X	X	E*		E*	E*
Direct Access (INPUT)	X				E	E
Direct Access (OUTPUT)		X		**	E	E
Direct Access (INOUT or OUTIN)	X	X			E	E
Direct Access (UPDAT)	X	X			X	

[1]To achieve card reader independence, CNTRL macro-instructions, if used, must be issued alternately with READ macro-instructions.
[2]The program is card punch type dependent if machine code control characters are used.
[3]The program is 1442 device dependent if WRITE is used.

The following operands must be specified in the FIND macro-instruction:

1. The name of the data control block of the input data set.

2. The address of an area containing either the name of the member, or the address of an entry for that member in a main storage list constructed by the programmer with a BLDL macro-instruction.

3. An indication of the type of information contained in the area parameter.

BLDL -- Build List

The BLDL macro-instruction causes member addresses and optional information from a data set directory to be placed into a specified list constructed by the programmer in main storage. The format of this list must be similar to that of the directory and must include the names of any members for which BLDL is to provide control information. Names must be placed into the list by the programmer in the same order in which they appear in the directory.

Access time for retrieval of members is reduced if a FIND macro-instruction is directed to an entry in a main storage list rather than to the directory. For this reason, the programmer may wish to construct such a list with BLDL for members frequently used during a program. A subsequent FIND macro-instruction for a member does not search the directory for a particular entry but simply refers to the list entry specified by the programmer.

The following operands must be specified in the BLDL macro-instruction:

1. The address of the data control block of an input data set.

2. The address of the main storage list to be built.

Note: It is only by use of BLDL that the programmer can retrieve optional information from the data set directory.

STOW -- Manipulate Partitioned Data Set Directory

The STOW macro-instruction causes the name of a member to be entered into or deleted from the data set directory with an indication of whether that member is an addition, a replacement, the same member with a new name, or a deletion. The physical position of the entry in the directory is determined by the name of the member.

The STOW macro-instruction permits more than one name to be entered into the directory for a given member. All names other than the first are called aliases. When a member is deleted or replaced, only the directory entry that is referred to is affected. Therefore, to produce consistent results, each alias for a given member must be deleted or replaced when that member is deleted or replaced.

The following operands must be specified in the STOW macro-instruction:

1. The address of the data control block of the output data set.

2. The address of an area containing the name of a member and, optionally, additional information for a directory entry.

3. An indication of the type of action to be taken on the member.

Note: It is only by use of STOW that the programmer can store optional information in the directory of a partitioned data set.

PROGRAMMING NOTE

Before issuing a FIND, BLDL or STOW macro-instruction, the programmer must test all preceding input/output operations for completion with CHECK macro-instructions.

CREATING A PARTITIONED DATA SET

A partitioned data set is useful for maintaining related groups of problem programs, often-used subroutines, and libraries of user-written macro-instruction definitions. The programmer creates a partitioned data set according to the following procedure:

1. Defines the data set using the DCB macro-instruction and the DD statement.

   a. Device must be direct access.

   b. Data organization field of the DCB macro-instruction must indicate partitioned organization.

c. Directory quantity subparameter must be specified in the SPACE parameter of the DD statement.

2. Issues the OPEN macro-instruction (output mode) to open the data control block of the data set to be created.

3. Uses the WRITE macro-instruction to place blocks on the direct-access volume.

4. When all the blocks that are to constitute a member have been written, issues a STOW macro-instruction to enter the name of the member, its relative address, and optional user data into the directory.

5. Continues to use the WRITE and STOW macro-instructions until all the members of the data set have been written, and their names have been entered into the directory.

6. Issues the CLOSE macro-instruction to close the data control block of the data set.

CONCATENATION OF PARTITIONED DATA SETS

Two or more partitioned data sets having identical characteristics can be concatenated for input processing as a single data set. The directories of the individual data sets of the concatenation are logically linked together and searched by the system (after FIND macro-instructions are issued) in the order in which the data sets are concatenated. If two members of different data sets have the same name, a FIND macro-instruction determines the address of the member whose name appears in the first directory searched. The programmer can use the BLDL macro-instruction and examine the list to identify the data set with which the member is associated.

The following considerations apply to concatenation of partitioned data sets:

1. Concatenation applies only to data sets opened for INPUT.

2. A maximum of 16 data sets can be concatenated.

3. All volumes containing the data sets to be concatenated must be mounted before program execution.

4. Only one data control block is associated with all the data sets concatenated. The programmer enters DD statements for these data sets in the order in which they are to be retrieved.

5. The end-of-data set routine is entered normally, i.e., when a block is requested and there are no more blocks in the member to be retrieved. The programmer may then choose either to process another member of the concatenated partitioned data sets or to issue a CLOSE macro-instruction.

BASIC PARTITIONED ACCESS METHOD COMPATIBILITY

The operating system provides limited compatibility between the basic partitioned access method and the sequential access methods. This compatibility enables the programmer to create, replace, or retrieve an individual member of a partitioned data set using the macro-instructions of either the queued or basic sequential access method. Use of the queued sequential access method in this manner is the only way blocking and deblocking of records is automatically provided in a partitioned organization. The factors governing these operations are outlined in the following text.

To create or replace a member of a partitioned data set using the sequential access methods, the programmer codes an OPEN macro-instruction (output mode) for the data control block of the data set to be processed, a series of PUT (or WRITE) macro-instructions, and a CLOSE macro-instruction. The data organization field of the DCB macro-instruction coded for this program should indicate sequential.

To create a new member of an existing partitioned data set, the programmer prepares a DD statement specifying, in the DSNAME parameter, the name of the data set and the name of the new member, and specifying MOD in the DISP parameter. When the data control block of this data set is closed, the name of the new member is automatically entered into the appropriate partitioned data set directory. An entry that has been automatically entered into a directory differs in no way from an entry that has been entered by the STOW macro-instruction, except that no optional user information may be included.

In a manner similar to that described in the preceding paragraph, the programmer can create a new partitioned data set with the data to be written as its first member. The programmer simply prepares a DD statement specifying, in the DSNAME parameter, the name of the data set and the name of the member of the new partitioned data set, specifying NEW in the DISP parameter,

and requesting the allocation of space for both the new data set and its directory. The data set and its directory are automatically created during execution of the program.

To replace a member of an existing partitioned data set, the programmer prepares a DD statement specifying the name of the member of the existing partitioned data set that is to be replaced, and specifying OLD in the DISP parameter. If the named member does not exist in the data set, the OLD specification is treated as if MOD had been specified.

To retrieve a member of an existing partitioned data set using the sequential access methods, the programmer issues an OPEN macro-instruction (input mode) for the data control block of the data set to be retrieved, a series of GET (or READ) macro-instructions, and a CLOSE macro-instruction. The data organization field of the DCB macro-instruction coded for this program should indicate sequential. The programmer prepares a DD statement specifying the name of the member to be retrieved, and specifying OLD in the DISP parameter. When the program is executed, the appropriate directory is searched and the address of the required member is automatically determined. The system, in effect, has issued a FIND macro-instruction.

Note: If the programmer does not specify a member name in the DD statement with the creation and retrieval operations described, the program is executed as for a sequential data set rather than as for a partitioned data set. Retrieval begins with the first block of the directory and terminates at the first partition encountered.

QUEUED INDEXED SEQUENTIAL ACCESS METHOD

The queued indexed sequential access method is provided to enable the programmer to sequentially store and retrieve the records of an indexed sequential data set. In addition, this access method is the sole means of creating an indexed sequential data set.

The macro-instructions of the queued indexed sequential access method, although similar in syntax and structure to those of the queued sequential access method, differ in function. These macro-instructions are completely direct-access device oriented, and automatically handle the intricate procedures involved in managing the index structure of the data set. Furthermore, most index searching is done by channel command sequences, without CPU intervention. The macro-instructions also provide automatic buffering and blocking/deblocking procedures, as required.

RECORD FORMATS

The queued indexed sequential access method supports four block configurations:

1. Format F, Unblocked.
2. Format F, Blocked.
3. Format V, Unblocked.
4. Format V, Blocked.

The formats of records as they appear on direct-access devices are shown in the following sections. Each block is recorded on a direct-access device with a count field, a key field, and a data field.

The key field is used by the system to locate a requested logical record. The data field contains the logical records.

Format F, Unblocked

Fixed-length unblocked records appear on direct-access devices as shown in Figure 16.

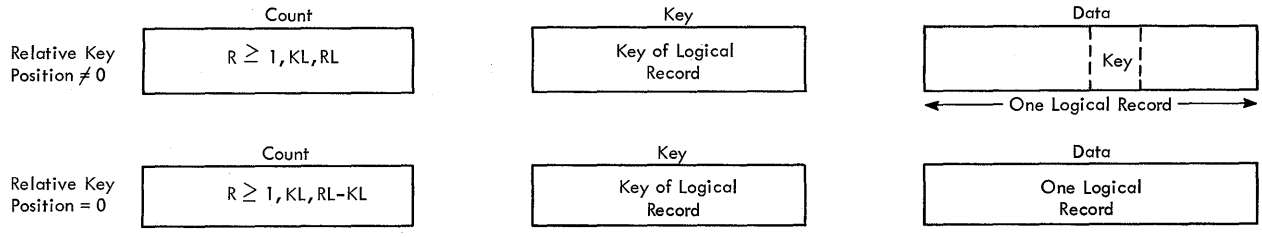


Figure 16. Unblocked Fixed-Length Records

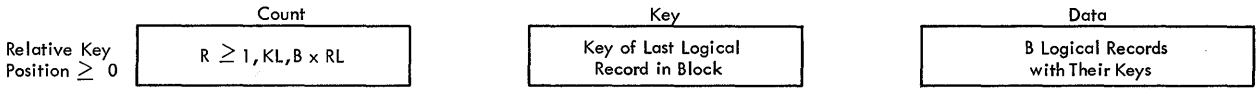


Figure 17. Blocked Fixed-Length Records

R
is the sequence number of the record on the track (R=0 is not used).

RL
is the length of the logical record including its key.

KL
is the key length. This length must remain constant for the data set.

Format F, Blocked

Fixed-length, blocked records appear on direct-access devices as shown in Figure 17.

R
is the sequence number of the record on the track (R=0 is not used).

RL
is the logical record length.

B
is the number of logical records appearing in a block and is a constant for a data set.

KL
is the key length. This length must remain constant for the data set.

More specifically, the key and the data areas may be pictorially represented as shown in Figure 18.

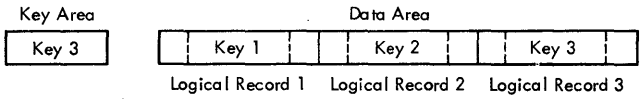


Figure 18. Key and Data Areas (Format F)

Keys 1, 2, and 3, respectively, are the keys of logical records 1, 2, and 3 and are physically embedded in the records. The position of the key in each logical record of the data set must be available to the system from either the DCB macro-instruction, the DD statement, or the data set label. The contents of the key field on the direct-access volume is used by the system for locating the block containing a requested logical record.

Format V, Unblocked

Variable-length unblocked records appear on direct-access devices as shown in Figure 19.

R
is the sequence number of the record on the track (R=0 is not used).

BL
is block length and includes the four bytes (LLbb) for the block length field.

KL
is the key length.

Format V, Blocked

Variable-length blocked records appear on direct-access devices as shown in Figure 20.

R
is the sequence number of the record on the track (R=0 is not used).

KL
is the key length.

BL
is block length and includes the four bytes (LLbb) for the block length field.

V
is the number of logical records that fit within the maximum block size specified for the data set. In general, V varies from block to block.

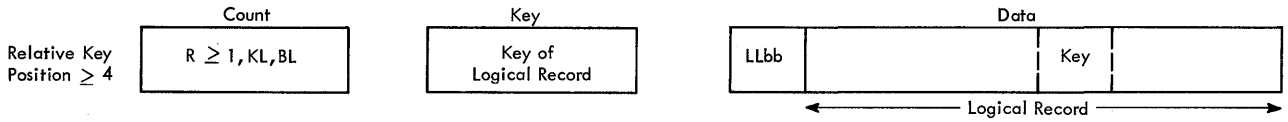


Figure 19. Unblocked Variable-Length Records

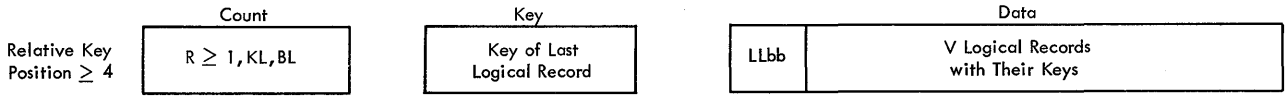


Figure 20. Blocked Variable-Length Records

More specifically, the key and data areas may be pictorially presented as shown in Figure 21.

Key 1 and key 2 are respectively the keys of logical records 1 and 2. The contents of the key field is used by the system to locate the block containing a requested logical record.

Overflow Records

The preceding figures are for prime (rather than overflow) data records only. Data records in an overflow area are organized somewhat differently, as shown in Figure 22. They are never blocked, even though the prime data records are blocked. In addition, they contain a link field which links to the next record in the overflow chain. The link field is 10 bytes in length.

Delete Codes

To mark a logical record for deletion, the user sets the "delete code" byte to all ones prior to issuing an output request for the logical record. For fixed-length formats, the delete code byte is the first byte of the logical record. For the variable-length formats, it is the fifth byte; that is, the first byte after the record control field.

The logical record is written in the prime or in the overflow area with its delete code. Physical deletion is performed if this record is forced off its prime data track by the insertion of a new record or if the data set is reorganized. The deleted record may be replaced by a record with an equal key with no error indication. Whether a record has been physically deleted or not, a GET macro-

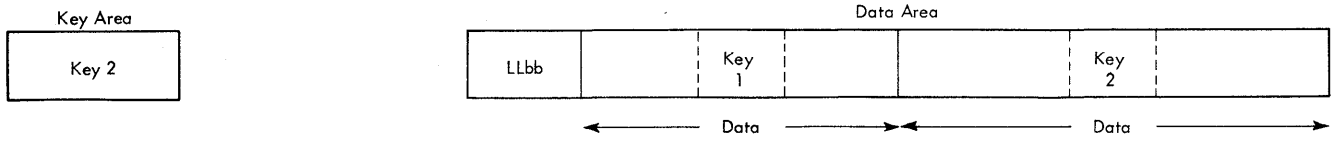


Figure 21. Key and Data Area (Format V)

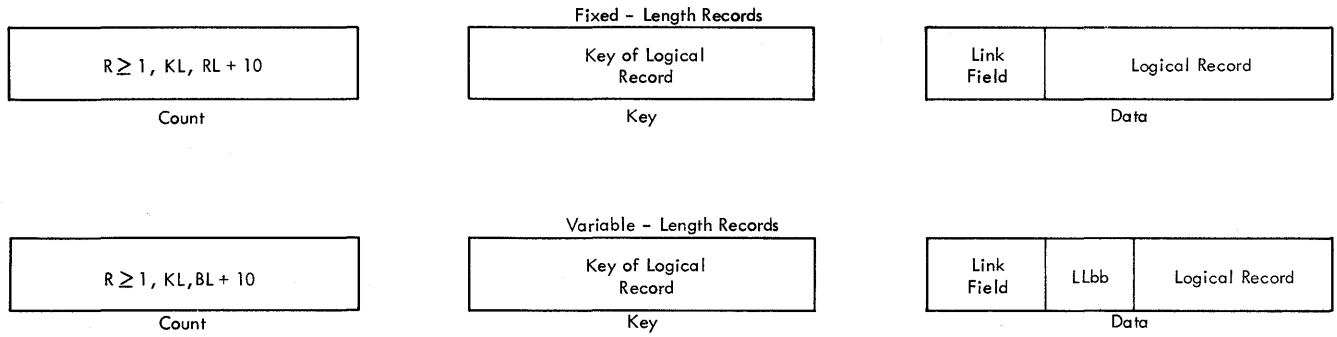


Figure 22. Overflow Records

instruction does not produce a record marked for deletion.

CAUTION: If the delete option is selected and the possibility exists that the first valid data byte of a logical record is all ones, e.g., a negative fixed-point number, the programmer should offset the beginning of his data from the delete code byte.

Also, if records are blocked and if relative key position is zero, a record marked for deletion cannot be replaced by one with an equal key, because the delete code will have overlaid the first byte of the key.

BUFFERING CONSIDERATIONS

The following section discusses the buffering facilities available with the queued indexed sequential access method. Descriptions of buffer pool construction and buffer assignment procedures are included.

Buffer Pool Construction

A buffer pool is automatically constructed for a data control block when it is opened if the programmer has not provided one prior to the conclusion of the optional data control block exit routine. The programmer must return this pool area to main storage when it is no longer required by issuing a FREEPOOL macro-instruction (refer to the section "Buffers and Buffer Pools").

When the programmer requires buffers for specialized applications, he should use either the BUILD or the GETPOOL macro-instruction (refer to the section "Buffers and Buffer Pools").

Buffer Assignment Procedures

After the buffer pool has been constructed, the programmer requests input or output of data records. The system automatically blocks and deblocks data records, obtains and controls buffers, and reads and writes data records, as required. It should be noted that only simple buffering may be used. If a buffer pool constructed by the BUILD macro-instruction is used by more than one data set, individual buffers from the pool are automatically assigned to the appropriate data control block according to the programmer's specifications.

MACRO-INSTRUCTIONS

The following macro-instructions are provided by the queued indexed sequential access method for input/output operations.

PUT -- Put a Logical Record

This macro-instruction is used only for creation of an indexed sequential data set.

The PUT macro-instruction causes a logical record to be placed into an output data set in either of two modes of operation: move or locate. In the move mode, PUT moves the logical record from an input buffer or work area into an output buffer segment. In the locate mode, PUT does not move the logical record into the output buffer, but places into a standard register the address of the buffer segment into which the programmer may move that record. The programmer specifies only one of these modes of operation as a parameter of the DCB macro-instruction associated with the output data set.

PUT stores logical records sequentially by keys. As required, PUT schedules the writing of output buffers, blocks records, creates all necessary indexes, and directs output error recovery procedures.

The following operands must be specified in the PUT macro-instruction:

1. The address of the data control block of the output data set.

2. The address of the output work area when in the move mode.

GET -- Get a Logical Record

The GET macro-instruction causes a logical record to be obtained from an input data set in either of two modes of operation: move or locate. In the move mode, GET moves the logical record from an input buffer into a specified work area. In the locate mode, GET leaves the logical record in the input buffer and places into a standard register the address of the buffer segment in which the programmer may process that record. The programmer specifies only one of these modes of operation as a parameter of the DCB macro-instruction associated with the input data set.

GET retrieves logical records sequentially by keys. As required, GET schedules the filling of input buffers, deblocks records, and directs input error recovery

procedures. After GET has successively retrieved all records to be processed, the system automatically passes control to the programmers's end-of-data set exit specified in the data control block.

For unblocked records, the programmer may specify in the DCB macro-instruction that device key fields are to be retrieved followed by the logical record. In the locate mode, GET places into a standard register the address of the record key. In the move mode, GET moves the key followed by the data of a logical record into the work area. For blocked records, a logical record is never retrieved with the key preceding the data, since key information is always embedded within each record. (See Figures 18 and 21.)

The following operands must be specified in the GET macro-instruction:

1. The address of the data control block of the input data set.
2. The address of the input work area when in the move mode.

RELSE -- Release an Input Buffer

The RELSE macro-instruction causes the GET macro-instruction to ignore the remaining logical records in an input buffer, and to obtain logical records from the next buffer. RELSE also causes the input buffer to become available for immediate refilling. When used with locate mode GET to refer to a data control block opened in UPDAT mode, RELSE prevents refilling of the buffer until its contents have been written, if PUTX was used. RELSE does not affect buffers containing unblocked records.

The address of the data control block of the input data set must be specified in the RELSE macro-instruction.

SETL -- Set Lower Limit of Scan

The SETL macro-instruction causes indexed sequential reference to an input data set to begin at any point in a data set with a record having a specified key or at a specified device address, and to continue sequentially by keys. SETL may also cause reference to begin at a specified key class. A key class is a group of keys sharing a common prefix of arbitrary length. The name of the class is specified by giving the prefix, followed by zeros, to the total key length. For example, a SETL with key class PE000 (in a data set consisting of the English dictionary) would cause sequential search to begin with PEA and continue to the end of the data set. Optionally, SETL also initiates retrieval of data area without keys. The SETL macro-instruction need not be used to initiate sequential reference. If a GET is issued and the data set is not already being sequentially referred to, sequential reference will be initiated at the beginning of the data set. For the unblocked formats, keys are read.

The following operands must be specified in the SETL macro-instruction:

1. The address of the data control block of the input data set.
2. The type of starting point for indexed sequential reference to the data set (either key, key class, device address, or beginning of data set).
3. Optionally, retrieval of data area without keys.
4. The address of an area containing either a key, generic key (key class), or device address.

ESETL -- End of Scan

The ESETL macro-instruction, by terminating anticipatory buffering, causes termination of indexed sequential reference to an input data set at any specified point in the data set.

The address of the data control block of the input data set must be specified in the ESETL macro-instruction.

Note: Reaching end of data set also terminates sequential reference.

PUTX -- Return a Logical Record

The PUTX macro-instruction causes a data set to be updated-in-place by writing back into the data set the same block that was previously read into a buffer. PUTX must be preceded by a GET macro-instruction that is in the locate mode. If PUTX is issued to any logical record or records in an input buffer, the entire buffer is written back directly to the data set after input processing of all the records in the buffer is completed. The programmer can mark a logical record for deletion by setting its delete code byte to all ones prior to

issuing a PUTX macro-instruction for the logical record.

The address of the data control block associated with the input/output data set must be specified in the PUTX macro-instruction.

EXCEPTIONAL CONDITIONS

The programmer can test for exceptional conditions and take appropriate action only by providing his own exceptional condition routine and specifying its address in the DCB macro-instruction of each data set to be processed. Failure to provide this routine results in termination of the job step when an exceptional condition occurs.

Table 5 describes, by macro-instruction, exceptional conditions that can occur with the queued indexed sequential access method. The exception codes that result from these conditions are described in the publication IBM System/360 Operating System: Control Program Services.

Table 5. Exceptional Conditions With the Queued Indexed Sequential Access Method

Macro-Instruction	Condition	Explanation
GET	Uncorrectable Error (Input)	System's standard error recovery procedures encountered an uncorrectable error in transferring block from secondary storage to the input buffer. System passes control to programmer's routine when GET is issued for the first logical record in the buffer containing block in error. The address of this buffer is placed into a standard register so that programmer can process the records in the buffer.
	Uncorrectable Error (Update)	System's standard error recovery procedures encountered an uncorrectable error in transferring a block updated-in-place (PUTX macro-instruction was used) from a buffer to secondary storage. System passes control to the programmer's routine when GET is next issued for the buffer containing the block that could not be written. The address of this buffer is placed into a standard register so that programmer can process its records. The system suspends buffer scheduling until a subsequent GET is issued.
	Unreachable Block (Input)	System's standard error recovery procedures encountered an uncorrectable error in searching an index, or the track containing the block sought; the next block of records cannot be located. When GET is issued for the first record of the block that could not be found, the system passes control to the programmer's routine. The programmer may decide to end sequential processing or terminate the job.
	Unreachable Block (Update)	System's standard error recovery procedures encountered an uncorrectable error in searching an index on the track containing the block to be updated. When a GET is issued for the first record in the buffer containing the block that could not be written, the system passes control to the programmer's routine.
SETL	Lower Key Limit Not Found	Key or key class that programmer specified as parameter of SETL is not found in the data set.

(Continued)

Table 5. Exceptional Conditions With the Queued Indexed Sequential Access Method (Cont.)

Macro-Instruction	Condition	Explanation
	Invalid Request	Either the programmer issued a SETL macro-instruction for a data set already being sequentially referred to by the same processing program, or the buffer cannot contain the key and the data, or the type specified in SETL has not been requested by the programmer in the data control block.
	Lower Device Address Limit Invalid	Device address that programmer specified in 'lower' parameter of SETL is outside the space allocated to the data set.
PUT	Space Not Found	Space allocated to the data set filled; no space to add records. The system passes control to the programmer's routine when such a PUT is received. In the locate mode, a buffer segment address is not provided by the system; in the move mode, the system does not move any data.
	Duplicate Key	Key of record to be transferred from main to secondary storage duplicates the key of a record previously addressed by PUT. The system does not transfer the record to secondary storage but passes control to the programmer's routine.
	Sequence Check	Numerical value of the key of the record to be transferred from main to secondary storage is less than that of the key of the record previously addressed by PUT. The system does not transfer the record to secondary storage but passes control to the programmer's routine.
	Uncorrectable Error (Output)	System's standard error recovery procedures encountered an uncorrectable error in transferring information to secondary storage. The system passes control to the programmer's routine when the next PUT is issued. If a data block could not be written, the address of the buffer containing the block in error is placed into a standard register. If the error is anything other than failure to write a data block, zero is placed into the standard register. After appropriate analysis, the programmer should close the data set or end the job. Subsequent PUT macro-instructions cause immediate reentry to the programmer's routine, since any attempt to continue loading the data set produces unpredictable results.

CREATING AN INDEXED SEQUENTIAL DATA SET

The programmer must follow the steps outlined below to create an indexed sequential data set:

1. Define the data set using the DCB macro-instruction and the DD statement.
   a. Device must be direct access.
   b. Data organization field of the DCB macro-instruction must indicate indexed sequential (IS).
   c. Macro-instruction reference field of the DCB macro-instruction must indicate only the PUT macro-instruction.
2. Issue the OPEN macro-instruction to open the data control block of the data set to be created.
3. Use the PUT macro-instruction to place all records or blocks of records that are to constitute the data set on the direct-access volume.
4. Issue the CLOSE macro-instruction to

close the data control block of the data set.

The records that constitute a newly created indexed sequential data set must be presented in ascending order by key. The programmer can merge two or more input data sets into an indexed sequential data set; for added flexibility, the system permits these data sets to be retrieved by any access method using any buffering technique provided by that access method. Once an indexed sequential data set has been created, its characteristics cannot be changed.

Note: PUT is the only macro-instruction that can be directed to an indexed sequential data set that is being created.

PROGRAMMING NOTES

The following section describes additional programming factors, not specifically related to any one macro-instruction, that the programmer should consider before using this access method.

Direct-Access Volume Option

The write validity check option, described previously, is available to the programmer.

Blocking of Variable-Length Records

When a PUT (locate mode) is used to block variable-length records, the system maintains a maximum record length in the data control block. When the space remaining in an output buffer is less than this maximum record length, the records in the buffer are transferred as a block to secondary storage and the system begins filling another buffer. To ensure that each block contains the maximum number of records it can accommodate, the programmer can replace the maximum record length maintained in the data control block with the actual length of each record about to be transferred to an output buffer.

BASIC INDEXED SEQUENTIAL ACCESS METHOD

The basic indexed sequential access method provides for direct storage and retrieval of the records of an indexed sequential data set.

The macro-instructions of this access method permit the direct retrieval of any logical record by its key, the direct update-in-place of any logical record, and the direct insertion of new logical records into an indexed sequential data set.

As an additional feature, the system provides, for this access method, a special buffering facility called the dynamic buffer option. This option enables the programmer to request system buffer management, and thus frees him of the responsibility for coding intricate buffer manipulation procedures. Furthermore, when the programmer specifies this option, the system defers buffer assignment until the actual data transfer is about to begin.

RECORD FORMATS

The basic indexed sequential access method supports four block configurations:

1. Format F, Unblocked.
2. Format F, Blocked.
3. Format V, Unblocked.
4. Format V, Blocked

For a discussion of record formats and other considerations refer to "Record Formats" in the section describing the queued indexed sequential access method.

BUFFERING CONSIDERATIONS

The following section discusses the buffering facilities available with the basic indexed sequential access method. Included are descriptions of buffer pool construction and buffer assignment procedures.

Buffer Pool Construction

The dynamic buffer option, enabling the programmer to request system buffer management, can be specified in the DCB macro-instruction. If this option is specified, the system obtains a buffer pool when the data control block of the data set to be processed is opened. This pool is automatically released when the data control block is closed.

Alternatively, the programmer can obtain a buffer pool by issuing either the BUILD or the GETPOOL macro-instruction (refer to the section "Buffers and Buffer Pools").

Buffer Assignment Procedures

To obtain a buffer from or to return a buffer to a pool constructed by either the BUILD or the GETPOOL macro-instruction, the programmer issues either a GETBUF or a FREEBUF macro-instruction, respectively. (See the section "Buffer Assignment Procedures" for the basic sequential access method.)

If the programmer has requested the dynamic buffer option, he obtains a buffer by requesting one in the READ macro-instruction. If the record for which the buffer is obtained is updated-in-place, the buffer is automatically returned to the pool when the programmer issues a WRITE macro-instruction with the appropriate parameter. If the record is not updated-in-place, the programmer must release each buffer that is no longer required by issuing the FREEDBUF (free dynamic buffer) macro-instruction.

FREEDBUF -- Free a Dynamically Obtained Buffer

The FREEDBUF macro-instruction causes a buffer that was dynamically obtained by the system to be returned to the pool from which it was obtained.

The following operands must be specified in the FREEDBUF macro-instruction:

1. The address of the data event control block associated with the READ macro-instruction that caused the buffer to be obtained.

2. A type parameter specifying the access method that is being used.

3. The address of the data control block of the data set.

MACRO-INSTRUCTIONS

The following macro-instructions are provided by the basic indexed sequential access method for input/output operations.

READ -- Retrieve a Logical Record

The READ macro-instruction causes the block containing a logical record with a specified key to be transmitted from an input data set to a main storage area. This area is either specified by the programmer or set up dynamically by the system at the programmer's request. Normally, the fields in the data event control block are specified as parameters of the READ macro-instruction (but may be subsequently replaced by parameters of other READ macro-instructions addressed to the same data event control block).

READ places into a standard register the address of the requested logical record. To allow overlap of the input operation with processing, READ does not wait for input completion but returns control to the problem program. The programmer issues a WAIT macro-instruction, following one or more READ requests, to delay processing until input operations are complete, after which the programmer may test the data event control block for errors and exceptional conditions.

A variation of the READ macro-instruction is provided for situations in which the programmer knows that the record being read will be returned to the data set. This variation, read for update, causes the physical location of the record just read to be retained by the system until a corresponding WRITE macro-instruction (referring to the same data event control block) is issued. Use of this technique increases performance rate by eliminating the need for the index search normally performed by the system when executing a WRITE macro-instruction. Additions to the data set are suspended until a WRITE macro-instruction has been received for each outstanding record read for update.

The following two operands must be specified in the READ macro-instruction:

1. The address of the data event control block (DECB) associated with the READ macro-instruction.

2. The type of input operation (normal reading; optionally, it can be read for update).

The following operands are required in READ macro-instructions that place the corresponding fields into a data event control block. In subsequent READ macro-instructions, they may optionally be specified to override the previously specified values. If an operand is omitted, the system assumes its value is already in the data event control block.

3. The address of the data control block of an input data set.

4. Either the address of an input area or an indication that the system is to obtain this area.

5. Either the number of bytes to be transmitted or an indication that the record length in the data control block is to be used. An overriding length must not exceed the length specified in the data control block of the input data set.

6. The address of an area containing the key of the record to be read.

Note: A buffer area must be large enough to accommodate the block containing the logical record.

WRITE -- Write a Logical Record

The WRITE macro-instruction requests that a logical record or a block be transmitted from a main storage area to an existing data set. WRITE may be used to update or replace existing records or to add new records.

Prior to issuing WRITE for updating or replacing a logical record within a block of such records, the programmer must issue a READ macro-instruction for that record. When replacing an unblocked record, the programmer need not issue a corresponding READ and must not attempt to replace the record with one of greater length. When adding a record, the programmer must be certain that the key of the new record is either unique or equal to the key of a record with an active delete code.

Normally, the fields in the data event control block are specified as parameters of WRITE (but may be subsequently replaced by parameters of other WRITE macro-instructions addressed to the same data event control block).

WRITE stores records by keys. To allow overlap of the output operation with processing, WRITE does not wait for output completion but returns control to the problem program. The programmer issues a WAIT macro-instruction to delay processing until output operations are complete. The programmer may then test the data event control block for errors and exceptional conditions.

The following two operands must be specified in the WRITE macro-instruction:

1. The address of the data event control block (DECB) associated with the WRITE macro-instruction.

2. The type of output operation (either replacement of existing records or addition of new records).

The following operands are required in WRITE macro-instructions that place the corresponding fields into a data event control block. In subsequent WRITE macro-instructions, they may optionally be specified to override the previously specified values. If an operand is omitted, the system assumes its value is already in the data event control block.

3. The address of the data control block of an output data set.

4. The address of an output area. If the output area is one that was obtained dynamically by the system in a previous READ macro-instruction, an indicator placed in this parameter causes the area to be returned to the pool after the WRITE completes. The corresponding field in the DECB must contain the address of the area.

5. Either the number of bytes to be transmitted or an indication that the record length in the data control block is to be used. An overriding length must not exceed the length specified in the data control block of the output data set.

6. The address of an area containing the key of the record to be written.

Notes: Since the basic indexed sequential access method has no facilities for creation of indexes, the WRITE macro-instruction cannot be used to place records into new output data sets but must be used with existing data sets. The queued indexed sequential access method macro-instruction PUT places records into new indexed sequential data sets.

A programmer can mark a logical record for deletion by setting its delete code byte to all ones prior to issuing the WRITE macro-instruction for the block containing the logical record.

EXCEPTIONAL CONDITIONS

When an exceptional condition that cannot be corrected by the system standard error recovery procedures results from a read or write operation, the system sets an exception code in the data event control block. The programmer is responsible for examining the data event control block for exceptional conditions, and taking appropriate action, if required. The format of the exception code and the exceptional conditions that can result from a read or write operation are described in the publication IBM System/360 Operating System: Control Program Services.

REORGANIZING AN INDEXED SEQUENTIAL DATA SET

As new records are added to an indexed sequential data set, overflow chains may be created. The access time for retrieving records in an overflow area is greater than that required for retrieving other records. Therefore, when many overflow records develop, input/output performance is sharply reduced. For this reason, the programmer should reorganize indexed sequential data sets as soon as the need becomes evident. The system maintains a set of statistics to assist the programmer in determining when reorganization is required.

These statistics are maintained as fields of the data control block, and can be tested by the programmer when he processes an indexed sequential data set. The field containing the number of tracks remaining in the independent overflow area assigned to the data set is automatically maintained. Two additional fields are maintained only at the request of the programmer. One contains the number of cylinder overflow areas that are full; the other indicates the number of times in a single processing of a data set using the basic indexed sequential access method that the programmer has gained access to an overflow record that is not the first in a chain of such records.

Reorganization is accomplished by creating a new version of the indexed sequential data set, using the existing data set as input for the data set to be created (refer to the section "Creating an Indexed Sequential Data Set").

PROGRAMMING NOTES

The following section describes additional programming factors, not specifically related to any one macro-instruction, that the programmer should consider before using this access method.

Overflow Records

When the system transfers a logical record to main storage from an overflow area, it places an indication of this transferral in the exception field of the data event control block associated with the READ macro-instruction. The programmer can test, after each WAIT macro-instruction that he issues for a read request, to determine if an overflow record has been read.

Direct-Access Volume Option

The write validity check option, described previously, is available to the programmer.

BASIC DIRECT ACCESS METHOD

The basic direct access method is the most flexible of those access methods that are provided specifically for use with direct-access devices. Using this access method, the programmer can directly store and retrieve a block by specifying either its actual device address, its relative position within a data set (relative block number), or the relative track within a data set at which the system is to begin a search. The system uses one of these specifications to locate the track containing a desired block. (Relative block numbers and relative track numbers are automatically converted into actual track addresses.)

After locating the proper track, the system must be directed to a particular block on the track. To indicate which block on the track is to be retrieved or stored, the programmer specifies either a block identification (actual position of the block on the track) or a block key.

The macro-instructions of this access method allow the programmer to easily perform such input/output operations as reading, writing, updating, and replacing the blocks of the data set.

As an additional feature, the system provides, for this access method, a special buffering facility called the dynamic buffer option. This option enables the programmer to request system buffer management, and thus frees him of the responsibility for coding intricate buffer manipulation procedures. Furthermore, when the programmer specifies this option, the system defers buffer assignment until the actual data transfer is about to begin.

In a multi-tasking environment, input/output operations of two or more independent tasks, as well as those of any one task, may be directed to the same data set. For this reason, it is possible that an input/output operation may affect the results of another input/output operation. For example, two tasks might simultaneously update the same block so that updating done by one task is destroyed by the other task. To prevent such occurrences, a special safety feature called exclusive control is

provided. Exclusive control can be requested by the programmer as a parameter of the READ macro-instruction.

The extended search option enables the programmer to request that the system extend its search for a record, or for space in which to add a record, beyond the relative track or relative record address specified in a READ or WRITE macro-instruction.

RECORD FORMATS

The basic direct access method supports five block configurations:

1. Format U.
2. Format F, Unblocked.
3. Format F, Blocked.
4. Format V, Unblocked.
5. Format V, Blocked.

All blocks are treated as the object of an input or an output request. Although the basic direct access method does not provide blocking/deblocking of records, the resources of the system permit the programmer to code these routines, if required, with a minimum of effort.

For format V blocks in the basic direct access method, the first eight bytes of a block (four containing block control information and four containing record control information) must be provided by the programmer.

BUFFERING CONSIDERATIONS

The following section discusses the buffering facilities available with the basic direct access method. Included are descriptions of buffer pool construction and buffer assignment procedures.

Buffer Pool Construction

The dynamic buffer option, enabling the programmer to request system buffer management, can be specified in the DCB macro-instruction. If this option is specified, the system obtains a buffer pool when the data control block of the data set to be processed is opened. This pool is automatically released when the data control block is closed.

Alternatively, the programmer can obtain a buffer pool by issuing either the BUILD or the GETPOOL macro-instruction (refer to the section "Buffers and Buffer Pools").

Buffer Assignment Procedures

To obtain a buffer from or to return a buffer to a pool constructed by either the BUILD or the GETPOOL macro-instruction, the programmer issues either a GETBUF or a FREEBUF macro-instruction, respectively. (See the section "Buffer Assignment Procedures" for the basic sequential access method.)

If the programmer has requested the dynamic buffer option, he obtains a buffer by requesting one in the appropriate parameter of the READ macro-instruction. If the block for which the buffer is obtained is updated-in-place, the buffer is automatically returned to the pool when the programmer issues a WRITE macro-instruction with the appropriate parameter. If the block is not updated-in-place, the programmer must release each buffer that is no longer required by issuing the FREEDBUF (free dynamic buffer) macro-instruction. (Refer to the section "Buffer Assignment Procedures" for the basic indexed sequential access method.)

MACRO-INSTRUCTIONS

The following macro-instructions are provided by the basic direct access method for input/output operations.

READ -- Read a Block

The READ macro-instruction requests that a block be transmitted from an input data set to a main storage area. The input area may be either specified by the programmer or set up and released by the system as required. Normally, the fields in the data event control block are specified as parameters of the initial READ macro-instruction, but may be replaced by parameters of subsequent READ macro-instructions addressed to the same data event control block. To allow overlap of the input operation with processing, READ does not wait for input completion, but returns control to the problem program. The programmer issues a WAIT macro-instruction, following one or more READ macro-instructions, to delay processing until input operations are completed.

The following two operands must be specified in the READ macro-instruction:

1. The address of the data event control block associated with the READ macro-instruction.

2. The type of block reference for input. This parameter indicates not only how the block is to be found, but also what portion of the block can be read into the input area. The block reference is specified as either block identification or block key. If block identification is specified, both the block key and the data portion of the block can be read (not necessarily into contiguous areas). If block key is specified, only the data portion of a block can be read. Optionally, exclusive control of input blocks or feedback of relative block addresses may be specified if these options are included in the data control block of the input data set.

The following parameters are required in READ macro-instructions that initially place the corresponding fields into a data event control block. In subsequent READ macro-instructions, the parameters may be optionally specified to override the previously specified values. When a parameter is omitted, the system assumes its value is already in the data event control block.

3. The address of the data control block of an input data set.

4. The address of an input area. If the dynamic buffer option is included in the data control block of the input data set, an indication that the system is to obtain the input area. (When control is returned from READ, the system leaves the address of the input area it obtained in the corresponding DECB field.)

5. Either the number of bytes to be transmitted or an indication that the block length provided by the data control block is to be used. An overriding length must not exceed the length specified in the data control block of the input data set.

6. If block reference is block key, the address of a field containing the key of the block to be read. If block reference is block identification, the address of a field into which the key is to be read. If the system is to obtain the input area, an indication that block key (followed by data) is to be read into this area. A value of zero for this parameter indicates that block keys are not to be read.

7. The address of the field containing either an actual or a relative address of an input block. If the feedback option is used, the system places the actual or relative address of the input block into this field.

Notes: The following notes apply to READ:

1. When exclusive control is specified, feedback of the actual address of blocks is automatically provided, unless relative track or relative block feedback is specified in the data control block.

2. When the type of block reference is block identification, the system does not provide the extended search option.

WRITE -- Write a Block

The WRITE macro-instruction requests that a block be transmitted from a main storage area to an output data set. Normally, the fields in the data event control block are specified as parameters of the WRITE macro-instruction but may be replaced by parameters of subsequent WRITE macro-instructions addressed to the same data event control block. To allow overlap of the output operation with processing, WRITE does not wait for completion of the operation, but returns control to the problem program. The programmer issues a WAIT macro-instruction to delay processing until output operations are completed.

The following two operands must be specified in the WRITE macro-instruction:

1. The address of the data event control block associated with the WRITE macro-instruction.

2. The type of block reference for output. This is specified as either block identification, block key, or an indication that new blocks are to be added. This parameter indicates not only how the system is to determine where the block is to be stored, but also how much of the block can be written. If block identification is specified, both block key and data can be written. If block key is specified, only the data portion of a block is written. If new undefined or variable-length blocks are to be added to an output data set, the count field of each block (provided by the system automatically) and the data portions are written. Writing the key field is optional, but if this option is

selected, all blocks must be written with keys. If new fixed-length blocks are added to an output data set, the count field is not written; however, the key field must be written.

The following parameters are required in WRITE macro-instructions that initially place the corresponding fields into a data event control block. In subsequent WRITE macro-instructions, the parameters may be optionally specified to override previously specified values. If a parameter is omitted, the system assumes its value is already in the data event control block.

3. The address of the data control block of an output data set.

4. The address of an output area. If the output area is one that was obtained dynamically by the system in a previous READ macro-instruction, an indicator placed in this parameter causes the area to be returned to the pool after the WRITE completes. The corresponding field in the DECB must contain the address of the area. Exclusive control of the updated block in this area, if specified, is also released at this time.

5. Either the number of bytes to be transmitted or an indication that the block length provided by the data control block is to be used. This overriding length must not exceed the length specified in the data control block of the output data set.

6. If block reference is block key, the address of a field containing the key of the block to be written. If block reference is block identification or if a block is to be added, the address of a field that contains the key to be written. A value of zero for this parameter indicates that block keys are not to be written.

7. The address of a field containing either an actual or a relative address of an output block. If the feedback option is used, the system destroys the value already in this field by placing the actual or relative address of the output block into this field.

Notes: The following notes apply to WRITE:

1. Exclusive control of blocks updated-in-place, if specified, is released after the output operation requested by WRITE is completed.

2. The programmer can write blocks on a track without knowing how much track space is available by indicating in the WRITE macro-instruction that new blocks are to be added. The block reference determines the track for the new block. When block format is specified in the data control block as V or U, the system determines, from the capacity record, whether the new block will fit. If the block fits, it is written following the last block currently on the track, and the capacity record is updated. For fixed-length blocks (which must have keys if the add feature is to be used), the system searches the track for a dummy block (indicated by all ones in the first byte of its key) and replaces the dummy block with the new block. For all block formats, the system sets an exception code when space for the new block is not found.

3. In the WRITE macro-instruction, the programmer must furnish keys of fixed-length blocks to be added, so that the dummy keys will be overwritten.

RELEX -- Release Exclusive Control

The RELEX macro-instruction causes an input block to be released from exclusive control, so that the block is available to other tasks requesting it. RELEX does not return control to the processing program until this action is completed.

The following operands must be specified in the RELEX macro-instruction:

1. The address of the data control block of the input data set.

2. The type of input operation (must be direct-access).

3. The address of a field containing feedback of block address.

Note: Blocks that are updated-in-place will be automatically released from exclusive control by the WRITE macro-instruction after the output operation is completed.

EXCEPTIONAL CONDITIONS

When an exceptional condition that cannot be corrected by the system standard error recovery procedures results from a read or write operation, the system sets an exception code in the data event control block. The programmer is responsible for examining the data event control block for exceptional conditions, and taking

appropriate remedial action, if required. The format of the "exception code" and the exceptional conditions that can result from a read or write operation are described in the publication IBM System/360 Operating System: Control Program Services.

EXCLUSIVE CONTROL

Exclusive control provides effective protection for a block only when exclusive control has been specified for all macro-instructions directed to the data set for updating operations.

When this condition has been met, the system permits only one input operation that specifies exclusive control to be performed on a given block at a given time. Other requests to read that block are deferred until the block has been released from exclusive control. For a block being updated-in-place, the programmer can release exclusive control by issuing a WRITE macro-instruction for that block. If a block is read and not updated-in-place, the programmer can use the RELEX macro-instruction to release control.

Exclusive control, although an optional feature, should be requested by programmers updating records in a multi-tasking environment. If only read operations are to be performed, exclusive control need not be requested. It should be noted that READ requests, for which exclusive control has not been specified, are not affected by the exclusive control requests of other macro-instructions. The programmer can therefore read a block that is concurrently being updated in exclusive status.

EXTENDED SEARCH OPTION

The extended search option enables the programmer to request that the system extend its search for a record, or for space in which to add a record, beyond the relative track or relative record address specified in a READ or WRITE macro-instruction.

To use the extended search option, the programmer must indicate in the DCB macro-instruction the number of tracks (including the starting track) or records (including the starting record) that are to be searched. If this number is 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 the programmer's request. If a request cannot be satisfied, the system sets an exceptional condition code in the appropriate data event control block. It should be noted that if the system cannot satisfy a search request after searching the highest relative track, it continues the search from the first relative track.

This option cannot be used if the type of record reference of the READ or WRITE macro-instruction is record identification, or if actual addressing of records is indicated in the data control block.

CREATING DIRECT DATA SETS

To create a direct data set, the programmer must use the basic sequential access method macro-instructions to set up capacity records and track formats of the direct-access volume assigned to the data set and, optionally, to write the data set blocks. Procedures for creating a direct data set differ according to block formats.

Format F Blocks With Keys

All tracks for a direct data set with fixed-length keyed blocks must be filled sequentially by either actual data set blocks or "dummy" blocks of the same length. The programmer issues the basic sequential access method macro-instruction WRITE to specify whether an actual or a dummy block is to be written. The system writes a dummy block with all ones in the first byte of the key field to indicate to the basic direct access method that a block can be added in the space allocated to the dummy block. (The programmer may not add actual data set blocks with keys whose first bytes are all ones.)

After the final block on a track is written, the basic sequential access method causes the capacity record to be written and proceeds to the next track. When a WRITE macro-instruction is issued after all tracks initially allocated to the data set are filled, the basic sequential access method causes additional space to be obtained and continues to fill tracks if secondary allocation was specified.

The programmer can determine whether the space initially allocated has been exhausted by checking the standard register that provides this information. The basic sequential access method also causes an indication of whether or not a track is full to be placed in the register unless

the track overflow option has been specified. This indication is useful, for example, when the programmer needs to know whether to complete a track with dummy blocks.

Format F Blocks Without Keys

The procedure for creating a direct data set with fixed-length blocks without keys is identical to that described for keyed blocks except that dummy blocks cannot be written. The programmer must fill all tracks for the data set with actual blocks by issuing the basic sequential access method macro-instruction WRITE.

Formats V and U Blocks

To create a direct data set with blocks of variable or undefined length, the programmer may write any desired blocks on the track by using the basic sequential access method macro-instruction WRITE. He then issues, for that track, a special form of the WRITE macro-instruction that causes the system to fill unused space with zeros and write the capacity record. (Throughout the remainder of this section, this form is referred to as the WRITE (capacity record) macro-instruction.) The next WRITE macro-instruction issued writes blocks on the following track.

When a WRITE macro-instruction is issued for a block that cannot be accommodated on the remaining track space, the block is not written. The programmer can determine whether the block was written by testing a standard register containing this indication. The programmer can then issue the WRITE (capacity record) macro-instruction, and reissue the WRITE macro-instruction to place the block on another track.

When a WRITE (capacity record) macro-instruction is issued as the first operation on a track, the entire track is cleared. When a WRITE (capacity record) macro-instruction is issued after all tracks initially allocated to the data set are filled, the basic sequential access method causes additional space to be obtained and continues to fill tracks if secondary allocation was specified.

Note: Since direct data sets are created using the macro-instructions of the basic sequential access method, the programmer can transfer direct data set blocks (in physical sequential order) from secondary to main storage, using the basic sequential access method READ macro-instruction. If the programmer uses this macro-instruction, neither capacity records nor the count area of the data blocks will be transferred; key areas, if requested and present, will be. All blocks, including dummy blocks, are transferred.

When creating a direct data set, each WRITE (capacity record) and WRITE macro-instruction must be followed by a CHECK macro-instruction.

PROGRAMMING NOTES

The following section describes additional programming factors, not specifically related to any one macro-instruction, that the programmer should consider before using this access method.

Split Cylinder Mode

Split cylinder space allocation cannot be used for a data set created for use with the basic direct access method, i.e., the data set cannot share cylinders with other data sets using this allocation technique.

Direct-Access Volume Options

The track overflow and write validity check options, described previously, are available to the programmer. The track overflow option, however, can be used only with format F blocks.

CUT ALONG LINE

READER'S COMMENTS

Title: IBM System/360 Operating System
Data Management

Form: C28-6537-1

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?
___ As an introduction to the subject ___ For additional knowledge
Other _____

fold

Please check the items that describe your position:

___ Customer personnel	___ Operator	___ Sales Representative
___ IBM personnel	___ Programmer	___ Systems Engineer
___ Manager	___ Customer Engineer	___ Trainee
___ Systems Analyst	___ Instructor	Other _____

Please check specific criticism(s), give page number(s), and explain below:
___ Clarification on page(s)
___ Addition on page(s)
___ Deletion on page(s)
___ Error on page(s)

Explanation:

fold

Name _____

Address _____

FOLD ON TWO LINES, STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

Form C28-6537-1

staple st

fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

fold

Printed in U.S.A. C28-6537-1

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

sta