File No. S370-30
Form No. GC28-2056-2

Systems Reference Library

IBM Time Sharing System

Data Management Facilities

This book is to be used as a reference guide for TSS users of data management facilities. Topics dealt with include: storage classes, unit record devices, data set characteristics, data set sharing, gaining access to data sets, and use of data management facilities. This book is equally useful to assembler, FORTRAN, or PL/I users.

The reader should be familiar with IBM Time Sharing System: Concepts and Facilities, GC28-2003.

PREFACE

This publication will provide users of TSS with an understanding of the data management facilities, and contains more than a "how to" description of these services. A working knowledge of the assembler language is required, particularly for understanding the description of the TSS access methods.

- Part I introduces such basic notions as that of a data set, in preparation for later discussions.

- Part II describes the manipulation and sharing of data sets within TSS; the notion of control blocks is introduced, and the system's access methods are discussed from the standpoint of the macro instructions which relate to each of them.

- Part III shows how such user-oriented facilities as the command system, and the FORTRAN language, make use of the system's data management services to serve a wide range of needs.

PREREQUISITE PUBLICATIONS

The reader must be familiar with the basic concepts and terminology of TSS, as described in IBM Time Sharing System: Concepts and Facilities, GC28-2003.

ASSOCIATED PUBLICATIONS

Other publications that will be useful for details not presented in this guide are:

IBM Time Sharing System:

Command System User's Guide, GC28-2001

Assembler User Macro Instructions, GC28-2004

System Programmer's Guide, GC28-2008

Assembler Programmer's Guide, GC28-2032

FORTRAN Programmer's Guide, GC28-2025

PL/I (F) Programmer's Guide, GC28-2049

Third Edition (December 1977)

This is a major revision of, and makes obsolete, GC28-2056-1.

This edition is current with Release 3.0 of IBM Time Sharing System/370 (TSS/370), and remains in effect for all subsequent versions or modifications of TSS unless otherwise noted. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters.

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

A form is provided at the back of this publication for reader's comments. If this form has been removed, comments may be addressed to IBM Corporation, Time Sharing System, Dept. 80M, 1133 Westchester Avenue, White Plains, New York 10604.

CONTENTS

Data Management is a general term that collectively describes the functions of the controlling system routines that provide access to data sets, enforce data storage conventions, and regulate the use of input/output devices. The data management facilities of TSS:

Permit the user to store, modify, and refer to programs and data, using the system storage facilities.

Free the user from concern with specific input/output device configurations.

Permit the user to defer such specifications as device type and length of records in the data set, until a program is submitted for execution.

Permit any desired interchange of programs and data among TSS installations.

Save the time and expense involved in writing routines similar to those provided by IBM.

Allow users to concentrate their programming efforts on processing the records read and written by the data management functions.

Provide standardized methods for handling a wide range of input/output and related operations.

Provide the flexibility for including new or improved devices, as they become available.

Provide comprehensive error-recovery procedures.

To most efficiently employ these facilities for his own purposes, the user of TSS should have a clear idea of those that are available to him, and a general idea of how they operate. This manual provides a sufficiently detailed description of the system's data management facilities to serve this need, without descending to a level of detail that would destroy the overall picture of these services.

Part I consists of descriptions of some introductory concepts necessary for a discussion of data management.

Part II describes how data management is effected in TSS, at its basic level.

Part III shows how higher-level, user-oriented services interact with the basic data management facilities, to perform a broader range of duties.

A record is a collection of related data items, treated as a unit. In data processing, a record is rarely considered or processed individually. Normally, records are treated in structured, logically related collections, called data sets. A data set may be, for example, a source program, a library of macro instructions, or a file of data records to be processed by a problem program. In general, data records are grouped as data sets because of some need to process them collectively.

NAMING AND CATALOGING DATA SETS

When a user wants to create or access a data set, he requires some means of specifying to the system the particular data set he is concerned with. Under TSS, the principal data management routines have been designed to free the user from considerations of data set residence, and delegate that responsibility to the system; also, many data sets are not physically connected entities, and may consist of widely scattered portions. For these reasons, data set specification by location cannot be the general rule; rather, the system must provide an interface that will relate the user's logical specification of a data set to that data set's physical location.

For direct access volumes, the system catalog serves to relate the user's specification of a data set -- the data set's name -- to a description of its physical structure, specified in a data set control block (DSCB). For tape volumes, the catalog links the name to the beginning of the data set on the appropriate volume. In designing the data set naming structure, one consideration was the sharing of data sets permitted under the TSS data management facilities. This sharing is implemented through the catalog; also, it is desirable to enable users to permit or restrict a sharer's access to a data set collection, rather than individually data set by data set. Example: If a user has a collection of data sets concerned with one project and wants to grant other users access to his entire collection, it will be easier for him and more efficient for the system to specify his group by one name. The user implies a data set hierarchy by specifying only a data set name; the specification of the upper portion of a hierarchy includes all the data sets logically below it.

Within TSS, a data set name is a series of one or more simple names, called components, joined so that each represents a level of qualification. Example: The data set name RECORDS.PERSONEL.DEPT561 consists of three components, delimited by periods; each component represents a unique category, within which the next component is a unique subcategory. In this example, some individuals might be permitted to access all records of the company, denoted by the partially qualified data set name RECORDS; others might be permitted to access all the personnel records of the company, denoted by the partially qualified data set name RECORDS.PERSONEL; some might be permitted access to only the personnel records of a particular department, RECORDS.PERSONEL.DEPT561.

A fully qualified data set name identifies an individual data set and includes all components of that data set's name. In the preceding example, the personnel records of Department 561 were uniquely identified by the fully qualified data set name RECORDS.PERSONEL.DEPT561. A partially qualified data set name identifies a group of data sets, and omits one or more of the right-most components of a data set name.

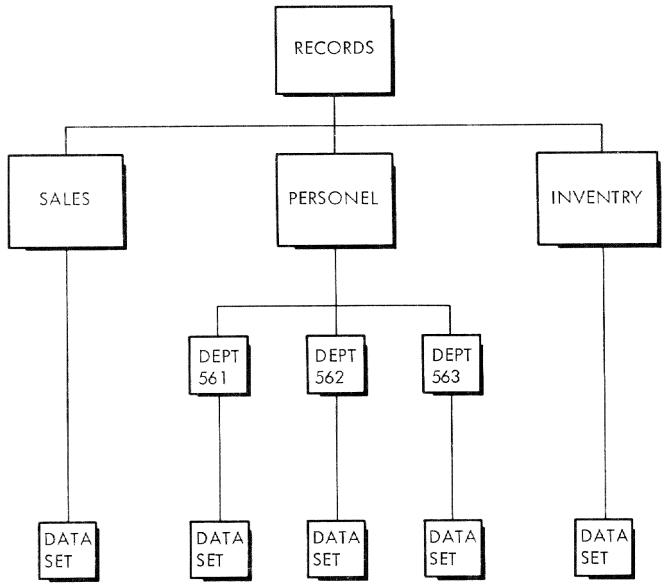


Figure 1. Fully and partially qualified names

In one example, all records of the company are designated by the partially qualified data set name RECORDS, and all the personnel records by the partially qualified data set name RECORDS.PERSONEL (see Figure 1).

These rules must be observed in naming data sets:

1. Each component except the last must consist of from one to eight alphameric characters (this is why "personnel", in the example and Figure 1, has only one N); the first character must be alphabetic.

2. The last component can consist of either alphameric characters, as in rule 1, or a relative generation number. A relative generation number consists of a signed integer in parentheses. Example: PAYROLL.CLERKS(-1). The system treats each relative generation number as the equivalent of an absolute generation number, which has the form GxxxxVyy, where xxxx is an unsigned four-digit integer and yy is an unsigned two-digit integer. Use of a generation number leaves a maximum of 26 characters available for higher-level qualification of the name. (Data sets cataloged under the same name but different generation numbers are generations of a generation data group, which avoids the necessity of giving a unique name to each data set. For more how-to information on generation data groups, see Concepts and Facilities, Command System User's Guide, Assembler Programmer's Guide, FORTRAN Programmer's Guide, or PL/I (F) Programmer's Guide.)

3. A period must be used to separate components.

4. For data sets used exclusively within TSS, the user is limited to 35 characters, because the system automatically prefixes each name with his eight character user ID, followed by a period. The maximum number of characters (including periods) in a data set name is 44. For data sets to be interchanged with the Operating System, the user can employ 44 character data set names. These data sets, however, cannot be cataloged in TSS without being renamed.

5. The maximum number of single-character qualification levels for a basic name is 18, for data sets used in TSS. Normally, fewer qualification levels will be used.

6. The fully qualified names in each user's data set name structure must be unique; no fully qualified data set name can be used as a partial qualifier for another fully qualified data set name.

Figure 2 illustrates how data sets on direct access volumes are located by data set name, under the system catalog concept. The system catalog is organized into a hierarchy of indexes, each index corresponding to a component of a data set's fully qualified name. The highest level index (the master index) is a set of user-identification codes, one for each user who has been granted access to the system. This master index is updated by the JOIN and QUIT commands given by the system administrators and manager. Each user ID

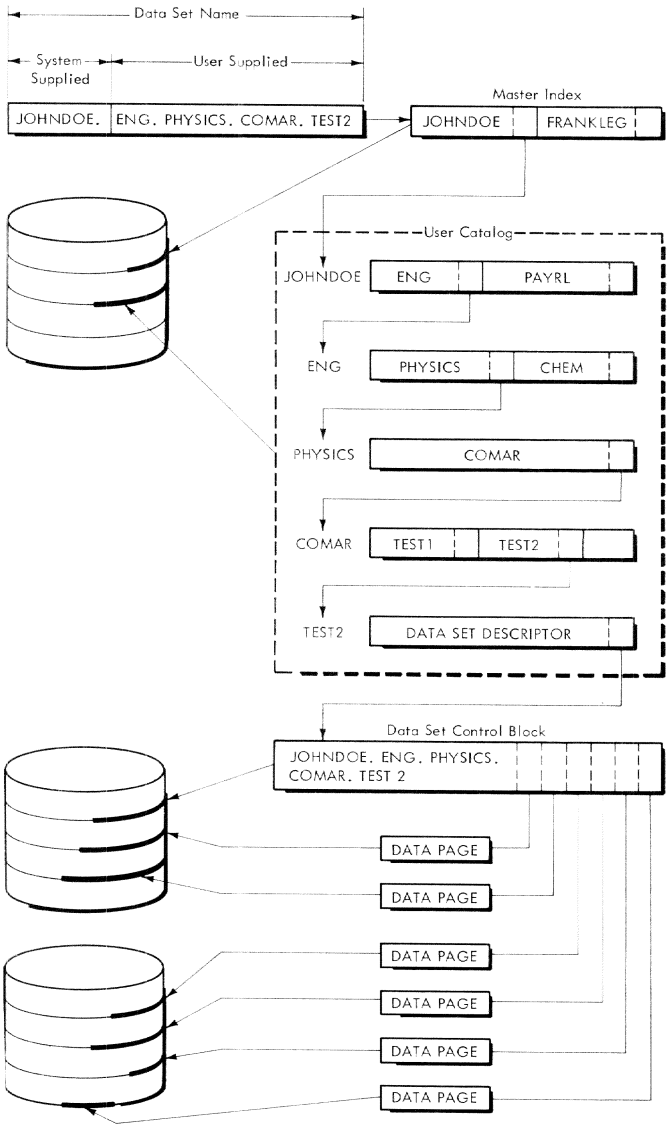


Figure 2. System Catalog Concept

in the master index points to a collection of indexes, called the user catalog. Each index in the user catalog corresponds to a level of qualification in the data set name structure adopted by the user. Users, therefore, determine the nature of their catalog by the way they name their data sets.

When a data set is cataloged, the required indexes are established in the user catalog in accordance with the fully qualified name of the data set. The lowest level index in the user catalog is called the data set descriptor and, for data sets on direct access volumes, it points to a data set control block (DSCB) that locates the individual pages of the data set. On magnetic tape volumes, the lowest level index of the user catalog gives the order, or sequence number, of that data set on a particular volume, relative to the beginning of the volume.

The explicit cataloging of data sets through the command system will be discussed in Part III. Data sets created by the usual TSS accessing facilities (the virtual access methods) are cataloged automatically when they are created.

UNIT RECORD DEVICES

A key concept in efficient device management depends upon the proper use of unit-record equipment (printers, card readers, and punches). By nature, this equipment can not be concurrently shared among several users; a unit-record device must be allocated to one job until it has completed using the device. Therefore normal users are not allowed complete control over unit-record devices in TSS because one user might tie up, for an excessive time, a piece of equipment that is needed by other users.

Most users obtain the services of unit record devices through the command system. Example: If a user wants to have a data set printed out, he issues a PRINT command; the system then initiates a batch job for the printing and thereby makes most efficient use of the printer.

Only users with privilege-class E (system monitors) can directly address specific unit record devices. In Part II, some of the access methods for this purpose will be described.

There are three storage categories: main, auxiliary, and external. Data is moved between main and auxiliary storage in a manner that is not evident to the user; these types of storage will be referred to only indirectly. External storage, however, is of more direct concern to the user.

PUBLIC AND PRIVATE STORAGE

The two types of external storage available to users are public and private.

Public external storage is that pool of storage available to be portioned out to users as they need it for creating or adding to their data sets. So that it will be a joint pool, capable of being apportioned efficiently, it must consist only of direct access storage. Direct access devices provide random access, so that previous positioning is irrelevant. Volumes that are not direct access -- for example, tapes -- cannot have control over them interspersed randomly among tasks, since one task would have no indication of where the previous one had positioned the volume. Therefore, only with direct access volumes can different portions of the same volume be efficiently parceled out to different tasks for immediate access. Hence, only direct access devices can be used for public storage; the system specifies the devices, and the volumes remain mounted throughout a session.

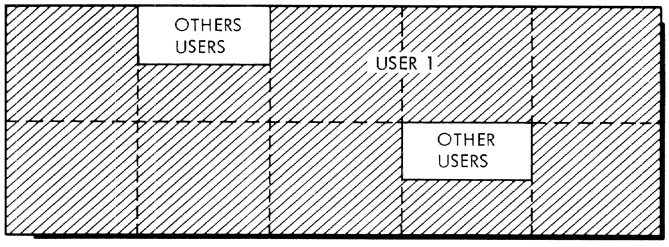
Private external storage is not a storage pool; it consists of volumes that may be allocated to only one task at a time. Because tape volumes can be allocated to only one task at a time, all tape volumes must be private storage. Also, direct access devices available to the system, which are not defined as public storage, are private devices that can be allocated to only one task at a time. Thus, private external storage may be either direct or sequential access volumes.

The system assumes that a user wants public storage unless he requests storage on a private volume. Public volumes are always mounted and available for allocation to a user's task.

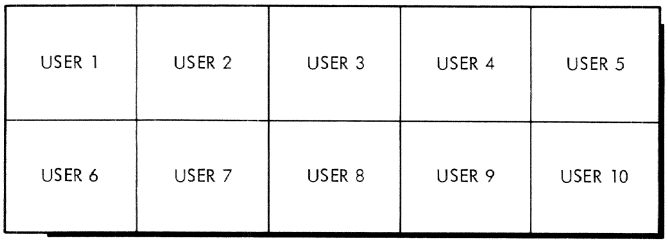
If a user wants private volumes, he may need to wait for devices on which to mount those volumes. Each time a request is made for a private-volume device, the system must determine if it can honor the request, based on the current availability of the device type specified, and the device ration permitted for the user. If no device is available to the task, a message is issued to the user (in conversational mode) or the system places the task in abeyance until the needed device can be allocated (in nonconversational mode). Conversational users can wait until a device is available, or perform other work.

PERMANENT AND TEMPORARY STORAGE

How can a public storage pool, available to all users, be rationed? One solution might be to give users as much as they want, whenever they need it. Unfortunately, this flexibility might lead to a single user severely reducing the public storage available to other users (see Figure 3). On the other hand, each user might have a maximum ration of 10% of the public storage. Besides being arbitrarily rigid (if only one person is using the system, why should he be so limited?), this solution limits the system to a maximum of 10 concurrent users (see Figure 3).



TOO FLEXIBLE



TOO RIGID

Figure 3. Public Storage Rationing

Public storage is rationed under TSS by restricting the extent that is allotted to any one user, but restricting it in a manner that allows considerable flexibility during the time span of a given task. Each user is allotted a maximum ration of permanent, and a maximum ration of temporary public storage. Data sets specified as permanent will continue to be on public storage after the task has logged-off; data sets specified as temporary will be erased automatically by the end of the task. Under this procedure, it is possible to allocate to a given user more than a strict percentage share of public storage, since it is known that the portion of storage that is occupied by data sets specified as "temporary" will be in use for only a short time. On the other hand, since even the extent of temporary storage available to a single user is limited by a fixed maximum, no one user can occupy an extensive area of public storage, even for a short time. Thus permanent- and temporary-storage rationing represents a compromise between the two situations depicted in Figure 3.

VIRTUAL STORAGE

Virtual storage is a name used to describe the logical storage space defined by the address capability of the TSS machine. Thus, for machines using a 24-bit address, virtual storage can represent over 16 million bytes, and for those using a 32-bit address, virtual storage contains over 4 billion logical (or conceptual) bytes. In TSS, each user has at his disposal the total amount of virtual or logical storage that is available to him based on his machines address capability (although some of this space may be used by system service routines that use virtual storage). The system provides this capability by translating virtual or logical addresses, specified by users, into actual storage addresses; all translation is transparent to the user. Users may program using virtual or logical addresses, and store data sets into virtual storage space; the system maps the program and data set addresses into actual storage addresses for him.

Three special Data Management Access Methods -- the Virtual Access Methods (VAM) -- have been provided with TSS. They are specifically designed for a time-sharing environment and are used to read and write data to and from direct access storage devices. For all three of the VAM access methods, the data set management (for example, formatting) is performed in virtual storage -- using virtual addresses that are part of the user's virtual storage address space -- although physical device management (e.g., I/O) is performed by system programs in resident storage. Each access method provides access and processing capabilities for data sets organized in a specific manner.

- sequentially (Virtual Sequential Access Method -- VSAM)
- indexed sequential (Virtual Indexed Sequential Access Method -- VISAM)
- partitioned (Virtual Partitioned Access Method -- VPAM)

In TSS, data sets organized for processing by one of the virtual access methods are generally referred to as VAM data sets.

VAM DATA SETS

VAM data sets reside on direct access external storage volumes, which are organized for maximum accessing efficiency. Volumes that contain VAM data sets can contain only VAM data sets; the formatting of these volumes is described in Appendix A.

The content of public storage consists exclusively of VAM data sets; direct access private storage, however, can contain either VAM data sets or data sets organized for interchange with any IBM Operating System (of these latter, only physical sequential data sets can be accessed under TSS -- see Appendix A for format descriptions). VAM data sets are not on tape volumes (although they may be temporarily stored as physical sequential data sets with the VT command, to be rebuilt in their original format by the TV command.)

Detailed descriptions of the various VAM data set organizations are given in Part II.

Note: Portions of VAM data sets do not reside in a user's virtual storage until they have been read in by one of the virtual access methods. Virtual storage includes the portions of any data sets that can be directly referenced (having been previously read), and may include data residing on main, auxiliary, or external storage (when the virtual access methods are used for reading).

We have seen that a data set is an organized, logically related collection of records. We will now briefly consider three basic ways that data sets are organized.

A distinction should first be made between two different types of records: logical and physical. A physical record is considered from the standpoint of the manner or form in which it is stored, retrieved, and moved -- that is, a record that is defined in terms of physical qualities. A logical record is considered independently of its physical environment -- more than one logical record may be within a single physical record.

Logical records are of primary concern to the user, since they are normally the units transferred to and from his problem program. Therefore, we shall refer to these simply as "records."

The concepts of logical and physical records lead naturally to record blocking; this is the combining of logical records into physical records that are to be transferred to or from an external device. Blocking will be described in Part II.

Records may be in one of three formats: fixed-length (format-F), variable-length (format-V or -D), or undefined (format-U). Formats are discussed fully in Appendix C.

Records are formatted, and the collections of records (that is, data sets) are also formatted. The format characteristics depend upon how they are to be accessed; generally they can be grouped into three categories: sequential, indexed, and partitioned.

Sequential Organization

When a data set is organized sequentially, the records are organized solely on the basis of successive physical positions, such as those of the records themselves, or of an associated pointer. One record precedes another logically if and only if it, or its associated pointer, precedes the other record physically. This implies how the data set is to be accessed (that is, how the records are to be read in and out). There is more discussion of this implication in Part II.

Indexed Organization

A data set with indexed organization has a unique key associated with each of its records. The key is a string that usually represents an item within the record, such as a part number, date, or name. The key may form the basis for accessing the records, or several keys may be ordered to permit sequential accessing. Accessing this form of data set is also discussed in Part II.

Partitioned Organization

A data set with partitioned organization has elements, or members, that are other data sets; these elements are in either sequential or indexed organization. A characteristic of a partitioned data set is a control directory that provides information about the location and number of the elements of the data set, and the characteristics of each element.

The organization within any of the three data set categories is dependent upon how the data set was created. Details of this dependence and data set organizations are in Part II.

Application: The names of the data set organizations were derived from combinations of the three categories and indicate characteristics to the user. Example: VIS is virtual index sequential organization; the V signals a virtual storage data set that must reside on a direct access volume, public or private. The next two letters, IS, provide progressively deeper levels of detail; "index" reflects that the logical order of the records of the data set is determined by a key; "sequential" indicates that the keys are arranged in an ascending collating sequence that provides an option of strict sequential access.

PART II: MANIPULATING AND SHARING DATA SETS

The characteristics of an individual data set must be clearly defined to the system before the user can create or access that data set by using the system's data management facilities.

Does the user want to write to the data set, read from it, or both? The system must have indications of how he intends to access a data set so that the routines that he will need will be available.

Does he want to provide his own routines for handling some processing interruptions? If he does, the system must have the addresses of these routines before it can start processing.

Such indicators are required at different times, and the sources of the indications differ: the user provides some and the system others. Control blocks act as storehouses for this information; the system will reference this information when it is required.

DCB AND JFCB

Most of the physical characteristics of a new data set are stored in the data control block (DCB), which normally resides in a user's program and is generated at assembly time. In addition to such static qualities as data set organization, the DCB contains information about processing requirements, among them the number of buffers required for input/output operations.

A particular DCB is not directly linked to any data set; rather, it is a "floating" definition that can be linked to a specific data set through an intermediate job file control block (JFCB), as shown in Figure 4.

The JFCB gives the user the flexibility to associate a DCB with a particular data set at the command-system level, anytime before executing a particular program. The user establishes the JFCB by issuing the DDEF command or macro instruction (or the CDD command, which copies prestored DDEF commands).

INTRODUCING A DATA SET TO A TASK

Besides creating a JFCB to link a data set and a DCB, the DDEF command (or macro instruction) introduces a data set (specified by its data set name) to a user's task. For a new private data set, the necessary messages are issued to the operator to request mounting of any required private volumes. If any volumes cannot be mounted, conversational users are informed by system messages; nonconversational tasks are either terminated by the operator or queued until the required private volumes have been mounted. (For nonconversational tasks, a list of private device requirements is made available to the system by the SECURE command, which the user must include in the task's nonconversational SYSIN as the first command after LOGON.) Then, as each DDEF is read and processed, the required devices are allocated.

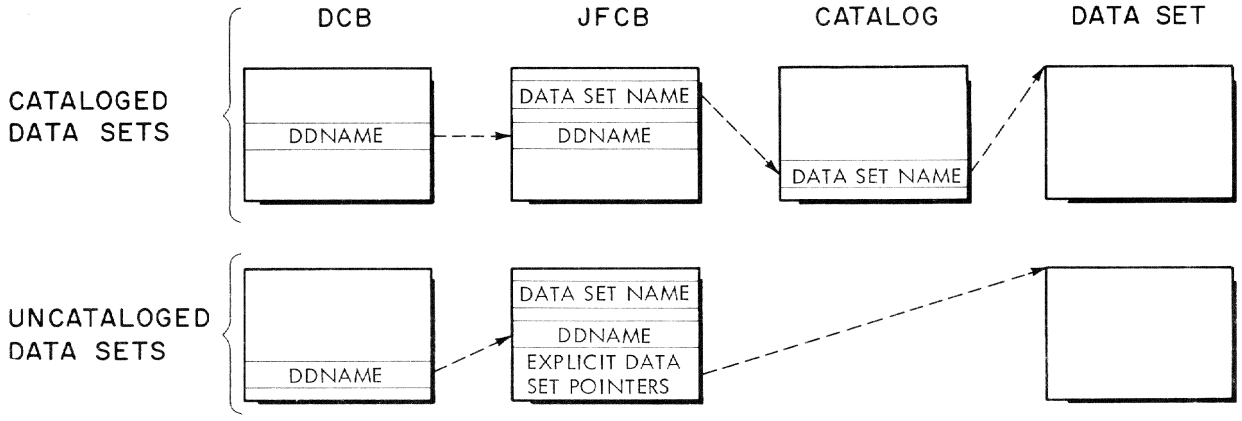


Figure 4. Data Set Description Control Blocks for Cataloged/Uncataloged Data Sets

For an existing data set with DISP=OLD explicitly stated, if a private (PS) data set is indicated the system requests mounting of appropriate volumes. If a public (VI, VS, or VP) data set is indicated, the system searches the catalog for the data set name and, if it does not find the name (that is, the data set is not cataloged), the system cancels the DDEF.

The catalog and DDEF values for data set organization, data set disposition, and device type must agree or the command (or macro instruction) is canceled. In all other cases of conflict, catalog information is used to fill out the JFCB, in preference to the conflicting information in the DDEF command or macro instruction. Since the user will not be informed of such conflicts, he should be aware of the JFCB fill hierarchy used.

For a new virtual storage data set (DISP=NEW), the DDEF command or macro instruction is canceled if the specified data set name already exists in the catalog.

Within any particular task, there can be only one JFCB for a data set. If a user issues a new DDEF with a data set name that is identical to that in a previous DDEF, a new JFCB will not be created. Instead, the ddname in the new DDEF will be substituted for the ddname currently in the JFCB, and processing for that DDEF will be considered complete. This will have the effect of associating a new DCB (the one associated with the ddname now in the JFCB) with the named data set.

The user can specify, in the DDEF, parameters for external storage space-allocation, device management, data set disposition, and DCB, that are to be put in the JFCB. The user may want to alter some parameters before executing a program, but issuing another DDEF with the same data set name will not affect any parameters in the JFCB, other than the ddname. If he issues the RELEASE command, the user can change parameters placed in the JFCB by a previous DDEF.

A DDEF command or macro instruction issued during a task is valid throughout the task, unless the user issues a RELEASE command for the data set named DDNAME in the DDEF. The RELEASE command releases the JFCB associated with a data set, removing the control information in the JFCB and disassociating the data set from a DCB. Releasing a data set does not uncatalog or erase it. If a private data set is released, and the volume on which it resides contains no other in-use data sets (i.e., data sets for which there are current JFCBs), then the I/O device on which that volume resides will be automatically freed for other uses.

Summary of DDEF Operands

Readers who require an in-depth treatment of the parameters available should consult Command System User's Guide or Assembler User Macro Instructions.

DDNAME: The symbolic data definition name that serves as the link between the DCB and the JFCB; since the JFCB, in turn, points to the data set, the DDNAME connects the data set attributes to a specific data set.

DSORG: Indicates the organization of the data set being defined; specification of this parameter must be correlated with the access method to be used in processing the data set.

DSNAME: Specifies the name under which the data set is to be cataloged or referred to for temporary reference; for data sets that are cataloged, this serves as the link between the JFCB and the data set.

DCB: Under this heading, parameters may be stored in the JFCB; these will be referred to when a data set is opened, for the purpose of filling out the DCB (see "Preparing Data Sets for Use").

UNIT: Specifies the type of device required by the data set. Direct access devices may be specified for either public or private volumes; other types of devices may be specified for private volumes only.

SPACE: Specifies the storage allocation for a data set that is to reside on direct access storage. Both primary and secondary space allocations can be specified; secondary space is to be allocated when the primary space has been filled.

LABEL: Applicable to data sets that are on tape, this operand specifies the sequence number (relative position) of a data set on a tape that contains multiple data sets. Also, this operand may specify the labeling conventions used with a data set; that is, if a data set is labeled and if the label is standard EBCDIC, standard ACSII, or user-created.

RETPD: Specifies the number of days the data set is to be retained by the system (retention period); applicable only to non-VAM data sets on direct access volumes or labeled tapes. When the specified time has elapsed, the volumes are available for reuse.

VOLUME: Specifies the volume on which the data set resides; this field must be specified when creating a new data set on a private volume or when referring to an uncataloged data set. It is also required when adding new volumes to an existing private data set. In general, it is required when the system cannot determine the necessary volume information from an existing catalog entry. Also, it may be specified for new data sets on public volumes, to restrict initial space allocation to these volumes.

DISP: Specifies whether the data set already exists or is to be created. If DISP is not specified, the system determines if the data set is new or old, based on whether there is a catalog entry for it; if yes, it is assumed to be old; if no, it is assumed to be new. If there is conflict between the specification of this operand and the state of a data set, the DDEF command is canceled.

OPTION: Specifies that either a job library is being defined or a data set is being added to the concatenated data set named in the DDNAME operand. A data set that is to be concatenated must exist in physical sequential organization. A job library that is being defined must be virtual partitioned; it will automatically be placed at the top of a list of job libraries defined during that task, and will be used to store object modules until either it is released, or a new job library is defined on top of it (job libraries are searched for object modules in the reverse of the order in which they were created).

RET: Specifies the storage attributes of a VAM data set. The user may specify permanent or temporary storage, that the data set is to be erased after the CLOSE or LOGOFF, or that either unlimited or read-only access to the data set is permitted.

PROTECT: Applicable to data sets on tape, this operand specifies whether file protection, that is, no file protect ring, is required.

PREPARING A DATA SET FOR USE

Even after a data set has been defined to a task and linked to a DCB by the creation of a JFCB, it is still not ready to be processed; the DCB may not be completely filled in, and the data set is not initially positioned. "Open processing" must be completed before the access methods can process the data set.

The assembler user initiates processing by issuing the OPEN macro instruction; for users of higher-level languages it will be automatically issued. When the data set is opened, the DCB is completed by filling in information obtained from:

1. Users' modification routines (BSAM only)
2. The DCB itself
3. The JFCB
4. The system catalog (for existing data sets)
5. Existing data set labels

Not all of these sources are valid for each field of the DCB. Two general rules apply: when a field is filled in by a higher priority source, it cannot be replaced by information from a lower priority source; if a field has not been specified by a higher priority source, it may be filled in by a lower priority source if that source is valid for that field. This flow of information is illustrated in Figure 5.

Open processing logically consists of two functions: a common portion that performs the services required by all access methods, and an access-method-dependent portion that completes the processing required by the pertinent access method.

The common portion first ensures that the DCB is valid; if not, the user's task will be abnormally terminated. Since the DDNAME parameter is required to provide the needed link between the DCB and the JFCB, before any filling in can begin a check is made to ensure that this parameter is in the DCB and that it corresponds with an identical parameter in some JFCB for that task. In case of any discrepancy, conversational tasks will be prompted and nonconversational tasks will be abnormally terminated. The user's authority code is checked to ensure that he is privileged to open this data set; if he is not, the task is abnormally terminated.

The access-method-dependent portion of OPEN completes any processing required for the device on which the data set is mounted, processes tape labels according to the open option specified (INPUT, OUTPUT, INOUT, OUTIN, RDBACK, or UPDAT), initially positions the data set, and sets up the linkages to the routines that may be used in accessing this data set. The routines that may be used depend upon the combination of the access method being used, the option with which the data set was opened, and the macro instruction references speci-

fied in the DCB. The initial positioning for non-VAM data sets is dependent upon the option with which the data set was opened, together with the DISP option of the DCB (OLD, NEW, or MOD).

Open processing for new VAM data sets includes creating catalog entries for these data sets; this occurs during common open.

Just as OPEN completes the logical connection between a data set and a DCB, permitting the data set to be accessed by the problem program, so CLOSE eliminates this link, removing the data set from direct contact with the problem program, and permitting it to be connected to a different DCB (or the same DCB, with different parameters). Then the data set can be accessed as a data set with different physical characteristics. (Note that this applies to the normal use of CLOSE; a more restricted form of the instruction, CLOSE (T), will be discussed below).

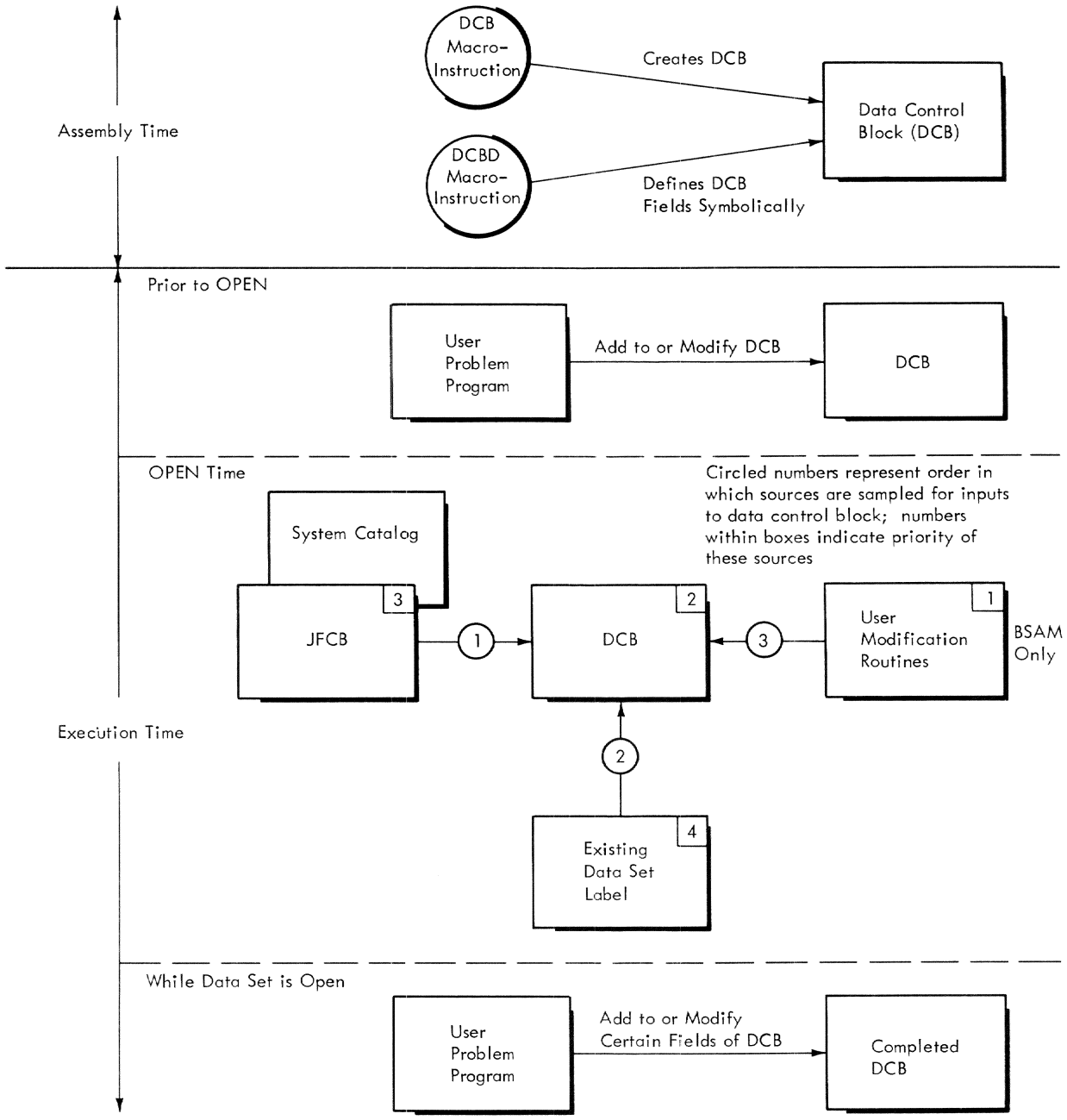


Figure 5. Flow of Information To and From a Data Control Block

Processing of CLOSE also consists of a common and an access-method-dependent portion. The common portion disconnects the DCB from the data set by returning the fields that were filled in during open processing to the condition they were in before the DCB was opened. After performing other processing, control is passed to the access-method-dependent portion, which checks all outstanding I/O operations for completion and then repositions the data set volume, if necessary (physical sequential data sets). Appropriate label processing is also performed; the type of processing is dependent upon the option with which the data set was opened. Example: a physical sequential data set that was opened for OUTPUT will have appropriate trailers written when it is closed.

A temporary close, CLOSE(T), can be performed if the user wants to reposition and consolidate status information, without disconnecting the data set from the problem program, by removing the link between the data set and the DCB. He may subsequently perform additional processing on that data set without again opening it. (For VP data sets, the FIND macro instruction must be issued prior to any further processing.) The temporary close performs the same processing as the standard CLOSE macro instruction, except that the fields of the DCB are not restored to the status they were in before opening. Also, if the user has specified the delete-at-close option in the DDEF for a data set, the standard CLOSE macro instruction will erase the data set before returning control to the user; CLOSE(T) will not.

The Duplexing Option

For public VAM data sets, an option provides for parallel creation and updating of two identical copies of a data set. With this option, when a user changes one copy of the data set, the system will automatically change the other copy.

The DUPOPEN macro instruction is used instead of OPEN when the user wants the system to maintain a copy of the data set. The user specifies, in the DUPOPEN macro instruction, the locations of the DCBs of the data sets to be maintained in parallel; in response, the system links the JFCBs associated with these data sets and allocates any necessary storage. Since the purpose of duplexing a data set is to provide protection against loss of virtual storage data sets through volume errors, external storage for each copy of the data set is allocated, wherever possible, on mutually exclusive physical volumes.

If an input error is detected on a page of the primary data set, the corresponding page of the secondary is obtained and used for input, and, also, is written back to overlay the logical page with the error on the primary data set. This process not only recovers from the error, but also tends to keep the primary copy in an error-free state.

To close duplex data sets, the DUPCLOSE macro instruction is used. To ensure that the two copies of the data sets are identical, the user must perform all operations on either data set within the duplexing mechanism (i.e., opened with DUPOPEN, and closed with DUPCLOSE). Errors will be indicated if the DCBs associated with duplexed data sets indicate conflicting attributes; similarly, the sharing properties specified in PERMIT commands (discussed in "Sharing Data Sets") must agree.

In many applications, more than one user may need the same data. Two or more users may want information from the same source of data or they may want to update the same copy of data. Enabling users to share one copy saves storage space; it also eliminates the need to collate information from different copies of the data to form a single updated copy.

Sharing may be external or internal. When external, several users have access to a copy that is contained on external storage; when a part of it is brought into a user's virtual storage, that becomes his private copy for his task alone. It is not affected by changes made to the external copy. When internal, the same copy is common to the virtual storage of all the sharers; a change made by one user is an immediate change to the copy used by all other sharers. Some system routines and control tables are shared internally; the sharing employed by most users is external.

EXTERNAL SHARING

The catalog is the mechanism by which data sets are shared externally. A user can allow any portion of his catalog to be shared; he can specify a data set that is to be shared, or he can specify an index level (a partially qualified data set name) to be shared. The latter will include all index levels below the shared index.

The user who authorizes sharing of a portion of his catalog is the "owner"; anyone authorized by the owner to share is a "sharer". The owner can specify the class of accessing privilege of the sharers of a data set or index level:

Unlimited access -- Sharers may read from the data set or modify it in any way; they may erase it.

Read-write access -- Sharers may read from the data set or modify it; they cannot erase it.

Read-only access -- Sharers may only read from the data set; they cannot modify or erase it.

Note: In the JFCB, there is a flag indication of whether a data set is sharable; issuing a PERMIT command does not alter an existing JFCB. Therefore, if a user decides to share a data set after he has already issued a DDEF for it, he should release the existing JFCB (by the RELEASE command or by logging off) and issue another DDEF for that data set, so that it will be flagged as sharable.

Through the PERMIT command, the owner can specify a data set or index level as universally sharable or he can explicitly specify the users who may share it; this information is placed in his catalog. Through the SHARE command, the sharer provides the linkage between his catalog and owner's catalog, and specifies this information:

Owner's ID,

Fully qualified name assigned by the owner to the data set or index level,

Fully qualified name assigned by the sharer.

Example: The owner, User 1, whose ID is USER1, specifies other users who may share index level A.B. User 2 wants to access User 1's data set A.B and call it X.Y. User 2 must then specify USER1.A.B and X.Y as parameters for the SHARE command. When User 2 wants to access data set X.Y a catalog search will be made through index levels X.Y.USER1.A.B to reach an entry in User 1's catalog (see Figure 6).

Sharing Private Storage

All data sets in a user's catalog, on either public or private storage, can be shared externally. However, since private volumes must be mounted on private devices, and private devices can be allocated to only one task at a time, a sharer may find that a private data set is unavailable to his task for one of two reasons:

1. There is not an appropriate private device available for allocation to his task.

2. Another sharer is currently using the data. (The private volume will not be available until he releases the JFCB or logs off.)

Sharing Public Storage

If the shared data is on public storage, the data set can be open and accessible to more than one task. When data is shared concurrently, records may be read and written by different users without any one having to close the DCB he has open for that

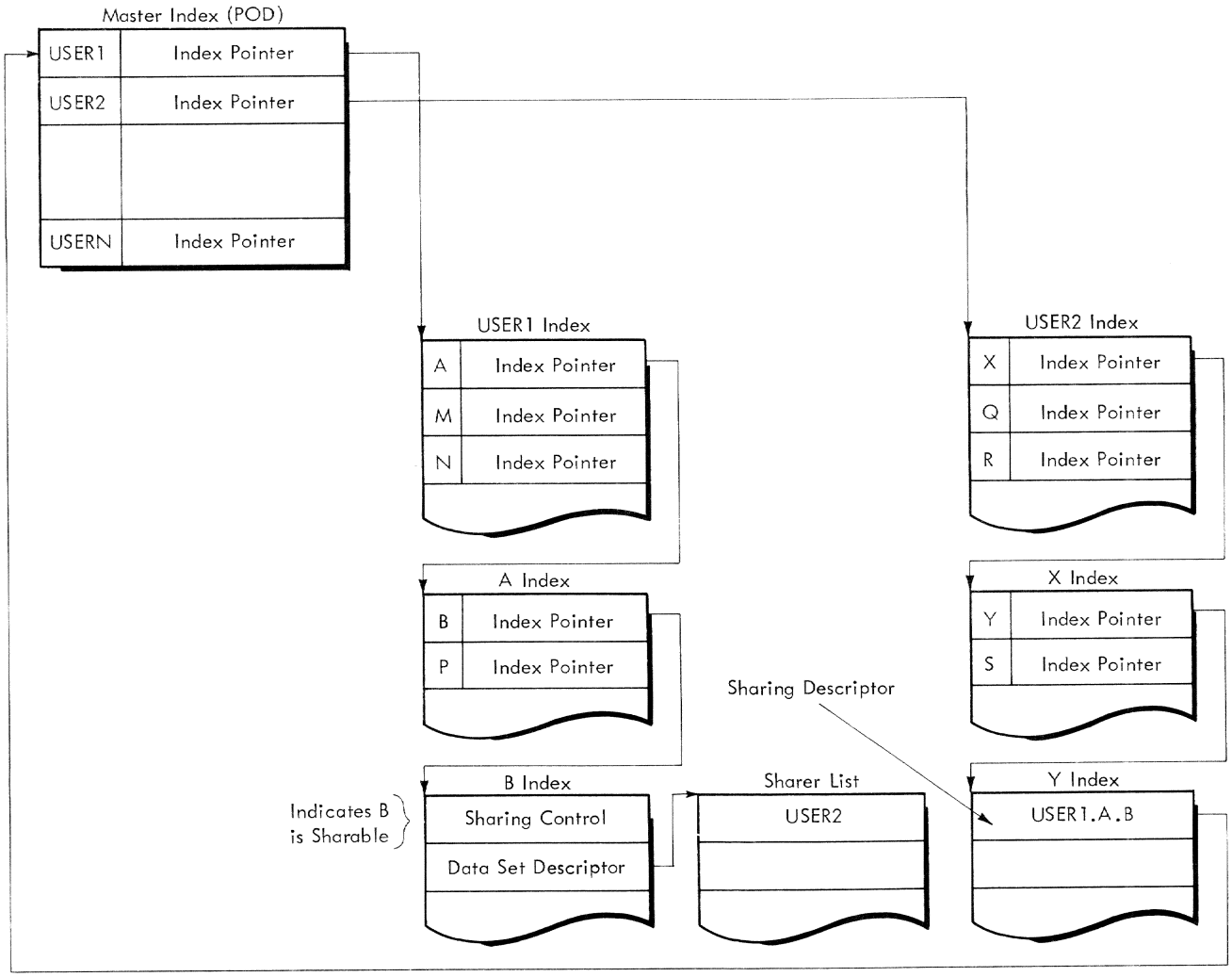


Figure 6. Example of External Sharing

data set. The sharer may have to wait until an interlock, set by another sharer's task, has been released, but this is the only restriction on the availability of the data. Two interlocks, read and write, control concurrently shared data.

A read interlock is imposed to prevent other users from writing into a data set, member, or page of a data set. Multiple read interlocks may be established for a data set or member, permitting several users to read it simultaneously, or the interlocks may be set on a page basis to give several users simultaneous access to the records within a page. A read interlock cannot be set if a write interlock has already been set for the data set or page. (For a VISAM data set, a data set level read interlock is slightly less restrictive; it prevents other users from opening that data set for output.)

A write interlock prevents any user, other than the user who set the interlock, from reading or writing into a data set, page, or member. Only one write interlock can be set at a time; thus, once a write interlock is set, neither read nor write interlocks can be applied until the write interlock is reset.

Data Set Interlocks: A data set interlock is set according to the option with which a data set was opened (INPUT, OUTPUT, INOUT, OUTIN, or UPDAT). It has the effect of restricting the OPEN options that will be accepted from future concurrent users; such users will be prevented from opening a data set with an option against which it is interlocked.

Member Interlocks: Partitioned data sets are interlocked at the member level, rather than the data set level; these interlocks

are set within the member header associated with the data set.

Page Interlocks: In shared VISAM data sets, interlocks are set at the page level; these interlocks are set by VISAM macro instructions.

Data set and member interlocks are released when the data set is closed, or the member is stowed; page interlocks are released when a reference is made to another page in the data set, when an ESETL or RELEX macro instruction is issued, or when the data set is closed.

INTERNAL SHARING

When virtual storage is shared internally, only one physical copy of the data is required in main storage, and is a part of the virtual storage of all the sharers.

One example of internal sharing is the shared data set table (SDST). It exists as part of the virtual storage that is initially allocated to all tasks when they execute LOGON, and is updated with entries for internally shared portions of shared data sets when the data sets are initially opened; users who subsequently open this data set reference this table, and increment the count of concurrent users in the table.

Internal sharing is effected by the dynamic loader for control sections with the PUBLIC attribute; the SDST serves as the link to this shared virtual storage.

The difference between internal and external sharing is illustrated by the system's treatment of PUBLIC and PRIVATE control sections in shared data sets:

User A wants to share module A with user B. Module A consists of two control sections, one with the PRIVATE attribute, the other with the PUBLIC attribute.

Since an object module exists as a member of a partitioned data set (let us call it JOBLIBA) and only entire data sets are shareable, User A must share data set JOBLIBA with User B. So User A issues a PERMIT command, naming JOBLIBA to be shared and User B as sharer. User B later issues a SHARE command, naming JOBLIBA. He decides to refer to this data set, for brevity, as LIBA and specifies this in his SHARE command.

User B now issues a DDEF for LIBA, specifing that it is to be a job library. Next, he requests the loading of module A from LIBA by calling that module.

But User A has already loaded module A and is presently executing it. The system will "connect" the PUBLIC control section already in shared virtual storage (it was loaded by User A) to User B's virtual storage; also, it will obtain a new copy of the PRIVATE control section from external storage and load it into User B's virtual storage. Thus User B is sharing module A with User A, although only one of the module's two control sections is being shared internally.

The system's access methods are at the heart of the data management routines. These are the techniques by which data is transferred between virtual and external storage. Since users must bring data into virtual storage to examine or process it internally, these access methods are constantly employed. Indirectly, they are used by FORTRAN, the command system, or a related method. General explanations of these interfaces are in Part III. Assembler users have a more direct link to these access methods through the macro instructions that they employ. The access methods are described, in terms of these macro instructions, in the sections that follow.

The access methods available to users fall into one of two catagories: the virtual access methods (VAM), and the sequential access methods (SAM).

The virtual access methods take maximum advantage of the time-shared environment of TSS and free the user from device considerations. When a VAM user creates a new VAM data set on public storage, the system allocates the needed storage from the pool of public volumes, and automatically catalogs it for the user. The data set may be spread across different public volumes, or even different device types, but it will be controlled as a logical entity for the user. VAM also uses the system's paging facilities for data transfer between external and main storage. So, the system can ensure efficient allocation of main storage by reading in only the pages of a data set that are being referenced during a particular time slice. This paging is not evident to the user and should not be confused with "reading into virtual storage" when a VAM user accesses a data record.

When a page containing a record is read into virtual storage, the necessary pointers are set up to "attach" this record to the user's virtual storage; the record's location will then be controlled by the system so that the user can directly reference the record when he needs it. When a reference is made to a record read by VAM, (whether the reference is made by the user or by a system routine), the system's paging facilities initiate any required I/O operations to ensure that the referenced page is brought in from external or auxiliary storage. This is the physical reading and may or may not be performed when the record is "read into virtual storage," upon the issuance of a VAM READ or GET instruction.

The sequential access methods provide a range of functions not available under VAM. Example: SAM users may access magnetic tape directly, or may access certain data sets created under the Operating System. However, the sequential access method MSAM is restricted to privilege-class E users and system routines; all it requires is that a user be authorized to access private volumes. Since SAM always requires the allocation of a particular private device to an individual user's task, SAM users will be forced to wait until the system can fulfill this need for a private device; this restriction does not apply to VAM users unless they are using private VAM. Finally, SAM can not take advantage of the system's paging facilities for record input from external storage; the SAM user must directly control the length of the physical record to be read into main storage. The SAM user's instruction, READ or GET, has a more immediate relation to actual data transfer than under VAM (that is, the input buffer will be filled as a result of the SAM macro instruction, not because of any subsequent reference to that record).

Users who access existing data sets are constrained (in terms of the access methods available to them) by the physical structure of the data sets. Example: Users who employ VAM must be sure that the data sets they are accessing are of VAM organization. Users who create new data sets must base their choice of access method upon the uses to which the data set will be put, as well as the system environment of TSS. Example: Users who want to store their data sets on tapes (perhaps to take advantage of the limited data set interchange with the Operating System) will employ the sequential access methods; users who want to take maximum advantage of the time-shared environment of TSS will employ VAM.

Within VAM or SAM, which access method a user should choose is determined by the manner in which he wants to access a data set. Example: When a data set, or a substantial portion of it, is to be processed sequentially, the GET and PUT macro instructions will often be the most efficient and convenient to use.

In determining the access method for creating a new data set, the user will in many cases be implicitly determining a great deal about the structure of that data set. Example: A user who selects VISAM to build a data set will automatically organize it as a VAM data set (see Appendix

A) and will restrict himself to the permissible record formats (F or V). Within the determined framework, the user can choose the record format that best fits the employment of the data set. A typical consideration might be processing speed. Since the access method must determine the record length for each individual record from the record itself, for format-V records, processing is slower than for format-F records, in which all lengths are the same. Therefore, when a data set will contain records of uniform lengths, or records that can, without much loss of space, be padded to uniform length, format-F is the most efficient.

VIRTUAL ACCESS METHODS -- VAM

The virtual access methods (VAM) are the principal means of data access in TSS. There are three virtual access methods, each of which provides access and processing capability for a specific type of data set organization:

Virtual sequential access method (VSAM)

Virtual index sequential access method (VISAM)

Virtual partitioned access method (VPAM)

VAM has been specifically tailored to make efficient use of the system resource of public storage space. To accomplish this, it is necessary to permit fragmentation of a user's data set within a public volume, across several public volumes, or even across several public-device types. This fragmentation prevents unnecessary gaps in public external storage. The efficient use of storage space by data set fragmentation is illustrated in Figure 7.

A data set may be fragmented, for example, when its size is being increased during different tasks. When a virtual storage data set is being created, a predetermined extent of external storage is allocated to it (this extent may be determined by user specification or system default). When the data set is closed, pages assigned to the data set that were not used will be freed (returned to the pool of available public storage). If, later, he increases the size of his data set, the additional external storage allocated may not be physically contiguous to the initial storage. The user is unaware that his external storage is not physically contiguous, since VAM organizes data sets by relative page number and logically

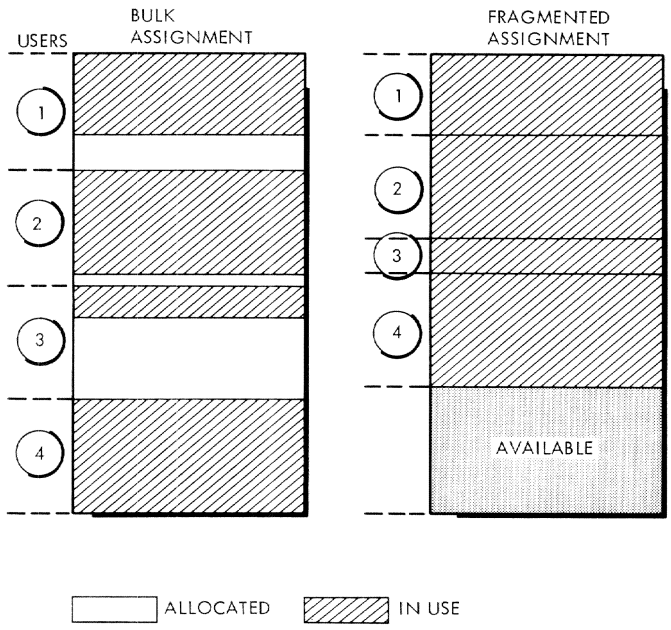


Figure 7. Bulk versus Fragmented Public Storage Assignment

connects the data pages, through the relative external storage correspondence table (RESTBL), which is in virtual storage. When data sets are opened, the system allocates virtual storage for the RESTBL; when they are closed, the virtual storage previously occupied by the RESTBL, is released and becomes available for system use (for shared data sets, the virtual storage for the RESTBL is only released when the last DCB for that data set is closed). On external storage, the information relating (a) the relative page numbers within the data set to (b) relative page numbers on the system's external storage is kept in data set control blocks (DSCBs). The DSCBs are used as source input to create the RESTBL and they are updated, when the data set is closed, to reflect changes to the data set.

The page-sized data blocks, into which virtual storage volumes are divided, are used by VAM as the unit of transfer between the direct access device and main storage.

The page-sized block for data storage was selected for a number of reasons. It is large enough so that direct access throughput is high, and the frequency of access requests by each user will be low. The direct access volume-packing efficiency is also quite high for page size blocks.

Processing Data Sets with VAM

Before a user can process a data set, he must DDEF it (directly or indirectly), and open the DCB associated with the data set

(or have it opened for him). A segment of open processing, known as open common, is basically the same for all data sets (and has already been described). The access method-dependent portion of processing that follows open common is determined by the data set organization. Initially, for all virtual storage data sets, this processing consists of building a RESTBL, performing some necessary duties for shared data sets, and then branching to one of two routines (depending on the virtual organization), which will make final preparations for the user's processing.

The overall concept of the virtual access methods, shown in Figure 8, includes the data transfers and logical relationships that occur when a user opens an existing VAM data set, uses VAM to request a logical record from it, and references that record for the first time.

When the user opens the data set initially, the information in the existing DSCBs is used by the OPEN routine to construct the RESTBL (1, in Figure 8). When he subsequently issues a locate-mode GET, the external storage address of the page containing the record is obtained from the RESTBL, and placed in an external page table (XPT) entry, which is associated with a virtual storage buffer (2, in Figure 8). Note that the external page containing the record is not read into main storage at this time. When the record in the virtual storage buffer is referenced, a paging relocation exception interruption occurs, and the paging mechanism proceeds to bring the page into main storage (3, in Figure 8). Thus VAM ensures that only the pages of a data set that are actually required for program execution are brought into main storage from external storage.

Virtual Sequential Access Method -- VSAM

The virtual sequential access method (VSAM) processes virtual sequential data sets and virtual sequential members of partitioned data sets. It can be used for any of these functions:

Create or extend a virtual sequential data set or virtual sequential member of a partitioned data set.

Delete all records in an existing data set or member from a specified record to the end of the data set or member (truncation).

Retrieve the logical records of the data set or member in a sequential or nonsequential manner.

Update, in place, an existing record of the data set or member.

To use VSAM to process a data set, that data set must have virtual sequential (VS) organization. As elements of a sequential data set, the records in a VS data set are ordered strictly by the sequence in which they were created. The user, in creating a VS data set, must provide the system with a stream of logical records that are concatenated and stored, page by page, on direct access devices. As each record is stored, the system makes its retrieval address available to the user's program. Users employing the assembler language can form another virtual sequential or virtual index sequential data set that contains these retrieval addresses. If the user wants to make an orderly sweep through the data set after he has created it, he can read the records back, in the order of creation, by requesting one logical record after another. An assembler user can also read and update logical records nonsequentially by specifying the required retrieval addresses of the records involved in SETL macro instructions; the retrieval addresses are in the data set that he formed.

All buffering required for VSAM processing (except for format-U move-mode, where the user's buffer is on a page boundary) is supplied by the system, based on the maximum logical-record length specified in the appropriate data control block. VSAM logical records may be format-F, -V, or -U. Record formats are described in Appendix C.

The macro instructions associated with virtual sequential data set processing are SETL, GET, PUT, and PUTX.

SETL specifies a logical record to be processed, using VSAM. The user needs to specify this macro only if he wants to process a record other than the next sequential one in a data set. SETL is called as a part of open processing, to initially position the data set for processing. If the DCB was opened for input, update, or in-out, SETL positions the data set at its logical beginning; if opened for output or out-in, the data set is positioned at the logical end.

GET obtains sequential access to a record of a VSAM data set. It may be specified by the assembler user in one of two forms:

Move Mode -- The user provides the system with the address to which he wants the record transferred; the system moves it.

Locate Mode -- The user requests the virtual buffer address of the next logical record in the input buffer in which the next logical record is

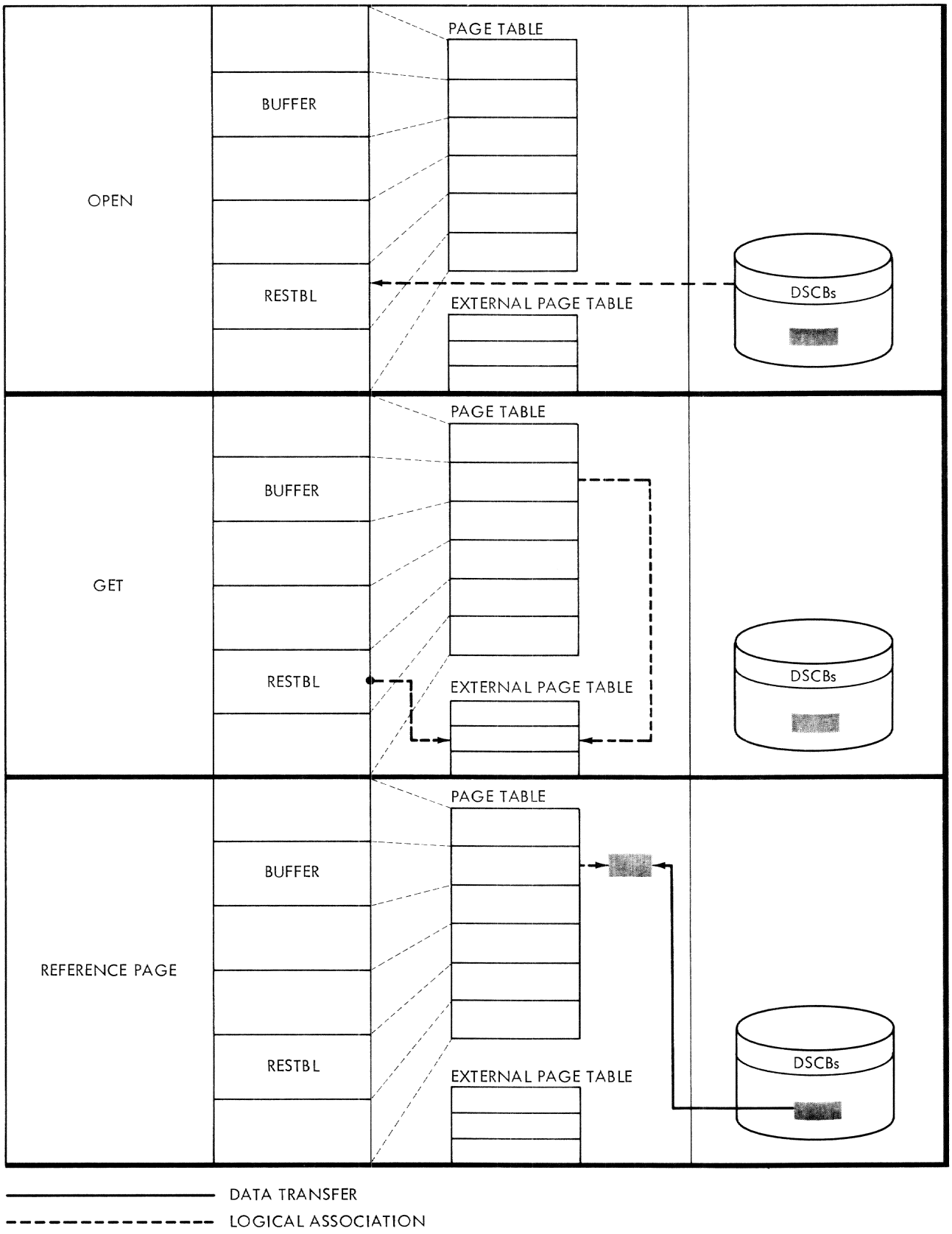


Figure 8. RESTBL, Virtual Memory, and Main Storage Relationships

stored. With this address, the user has the option of processing the record at that location or moving it to his own work area. In "locate mode," there is no actual record transfer until the user references the record and a page relocation interruption occurs; this is also true for format-U records in "move-mode."

In processing format-U records, the user must specify their lengths in the data control block prior to issuing the GET macro instruction; VSAM format-U records must be even-multiples of a page.

After each execution of the GET macro instruction (in either mode), the retrieval address of the logical record just retrieved is in a data control block field. The user may create a secondary data set from all his GET retrieval addresses to facilitate future nonsequential processing of the original data set.

Successive GET macro instructions will retrieve the records of a data set in the sequence of creation. When the system detects the end-of-data condition while processing a GET instruction, the system will transfer control to the user's end-of-data (EODAD) routine. To start sequential processing at a point other than the beginning of a data set, the user can specify the retrieval address in a SETL macro instruction, prior to issuing the GET macro instruction.

Similarly, to directly access any record in the data set when its data control block is open, the user can specify its retrieval address in a SETL macro instruction and then issue a GET macro instruction to obtain it.

PUT places logical records into an output data set when a virtual sequential data set or member is being created or added to. In addition to concatenating records into a data set, PUT defines a new end-of-data set to the system each time it is issued. It may be issued by the assembler user in either of two forms:

Move Mode -- The user provides the system with the address of a logical record; the system transfers the record from that location to the next available output buffer segment. From there, the system automatically writes the record to the output data set before that portion of the buffer is released or reused.

Locate Mode -- The user requests, from the system, the address of the next available output buffer segment. He uses that address to store the logical record that he wants to add to the data set; the system automatically writes the record to the output data set when necessary.

The user must specify the length of the logical record for each PUT macro instruction. For format-F records, this information is in the data control block and is the same for each record in the data set. For format-U records, the user stores this information in the data control block prior to each PUT. For format-V records, the lengths must be supplied within each logical record by the user. The length of each logical record must not exceed the maximum specified in the data control block at open time.

When a PUT macro instruction is issued in either mode , the retrieval address of the record to be stored is made available by the system (in a data control block field). The user can store these addresses, and use them in the SETL macro instruction for later nonsequential processing of the data set.

The PUT macro instruction may also be used to truncate an existing data set. Since the system automatically generates an end-of-data indicator as part of the execution of every PUT, the user could issue a SETL instruction to position a volume at, say, the middle of a data set, and then issue a PUT for a certain logical record; the system will then indicate that the record is the new end of the data set. The records that were previously in the last half of the data set have now been deleted.

PUTX rewrites an updated logical record from an input buffer area, back to a data set on external storage; the record must have been brought from external storage to the buffer area by the execution of a locate-mode GET instruction. If the user attempts to change the length of the record he is updating, or if the DCB associated with that data set was not opened for update, the user's task will be abnormally terminated.

VSAM Sharing Controls: The system provides interlocks for shared virtual sequential data sets: If a VS data set is opened for input, other users can read the data set, but they cannot write into it; if a VS data set is opened for output, update, in-out, or out-in, no other user may have any access to that data set; the data set cannot be opened for these options if anyone else is using it. All interlocks are automatically removed when a data set is closed.

Virtual Index Sequential Access Method -- VISAM

The virtual index sequential access method (VISAM) processes virtual index sequential data sets and virtual index sequential members of partitioned data sets. It can be used for any of these functions:

Create a virtual index sequential data set or member, in a sequential or nonsequential manner.

Retrieve the logical records of the data set or member, in a sequential or nonsequential manner.

Update records in a sequential or nonsequential manner.

Insert new records in their logical sequence within the data set or member.

Delete selected records from the data set or member.

To use VISAM for data set processing, the data set must have virtual index sequential (VIS) organization. As elements of an indexed data set, the logical records of a VIS data set are organized in an ascending collating sequence, based on a unique data key associated with each record. The data key may be a control field that is a part of the record (such as a part number), or it may be an arbitrary identifier (such as a line number) that is added to each logical record.

In each page of the data set there is an ordered set of locators, one locator per record. Each locator specifies the physical location of the record on the page. Locators are placed sequentially (lowest key first) at the bottom of the data page in ascending order, the locator for the lowest key on the page is at location X'FFC' into the page. Location X'FFE' contains a half word displacement to the end of the locators (highest record on the page) and is adjusted upwards and downwards as records are added and deleted. Records may or may not be logically sequential on a page, locators are always in sequential order.

Even though a VI dataset is logically sequential its physical pages may or may not be. Control of this processing is maintained by using the VISAM directory as a translating mechanism to convert logical pages (and their records) into the actual physical location of the page which contains the desired record. The directory for a VI dataset is built and maintaind by the VISAM access method routines after the number of data pages in the dataset exceed one. There is one key entry in the directory for each data page in the dataset except page 1 (PPN 0). The key entry contains the logical position of a page in the dataset as well as its physical location. Key entries are in the following format:

Bytes

0-1	logical page number (LPN) this key entry
2-3	physical page number (PPN) this key; location of this page relative to 1st data page of this dataset
4-5	old physical page number (OPPN) on the page; PPM value on page before its physicall page number relative to the dataset was changed
6-9	spare
A-N	low key on this page, rounded to a half word boundry

The 1st 2 bytes of each data page contain the PPN of the page. A page's PPN number will always equal the PPN value in the directory unless there have been some pages deleted, in which case the old page PPN is saved in the key entry. This PPN number is used for validity checking of VISAM pages by the VISAM input page routine to ensure dataset integrity. New pages are always added to the end of the dataset even though they may logically represent an insertion somewhere in the middle. By adding pages at the end and maintaining a translation mechanism the need for overflow pages is eliminated.

Insertions (records) are added to an existing full data page according to the following rules:

A. If the new record to be added to the dataset is going to be the last record on the page, a new page is added to the end of the dataset, and the new record (key) will become the low key on the new page. The key entry for the new page will be inserted in the correct logical position in the directory.

B. If the new record is not the last record on the page all records with a key value greater than the new record will be moved to a new page and a new key entry added to reflect the new page. The new record may or may not fit on the old page. If it does not proceed with (a) above. (See Figures 9A - 9C.)

VISAM DIRECTORY

header				
LPN	PPN	OPPN	SP	KEY
0001	0001			KEY300
0002	0002			KEY500

DATA PAGES

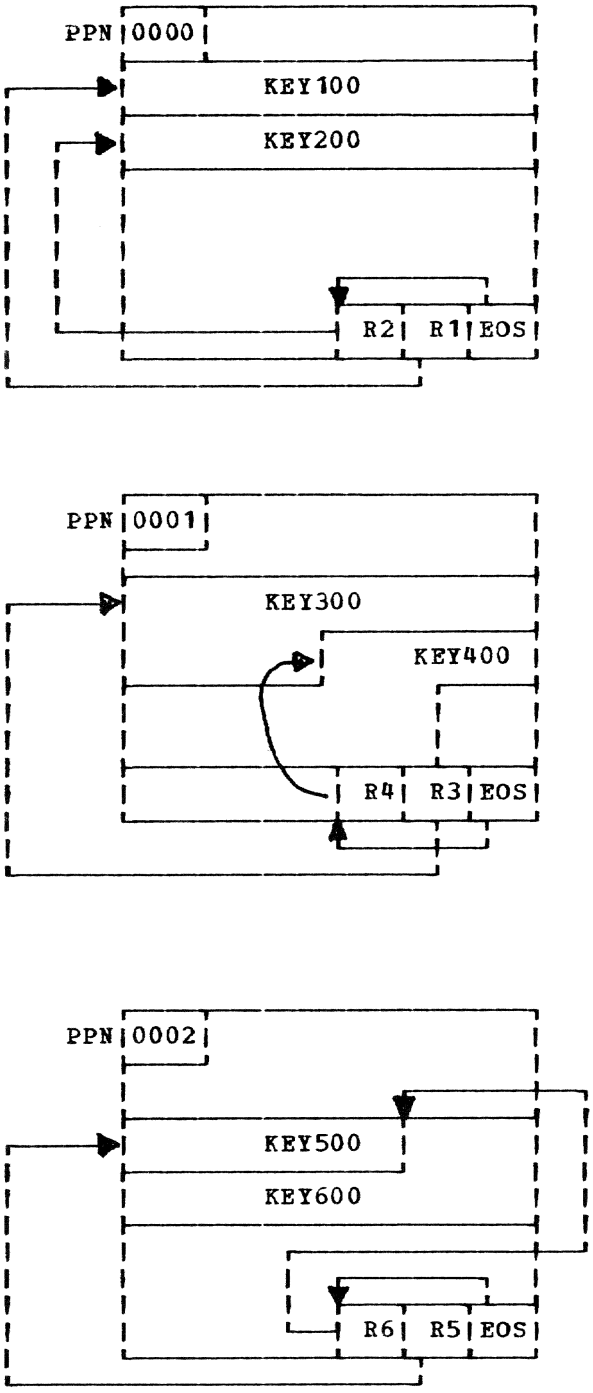


Figure 9A. Typical 6-record VISAM dataset created sequentially

VISAM DIRECTORY

HEADER				
LPN	PPN	OPPN	SP	KEY
0001	0001			KEY300
0002	0003			KEY450
0003	0002			KEY500

DATA PAGES

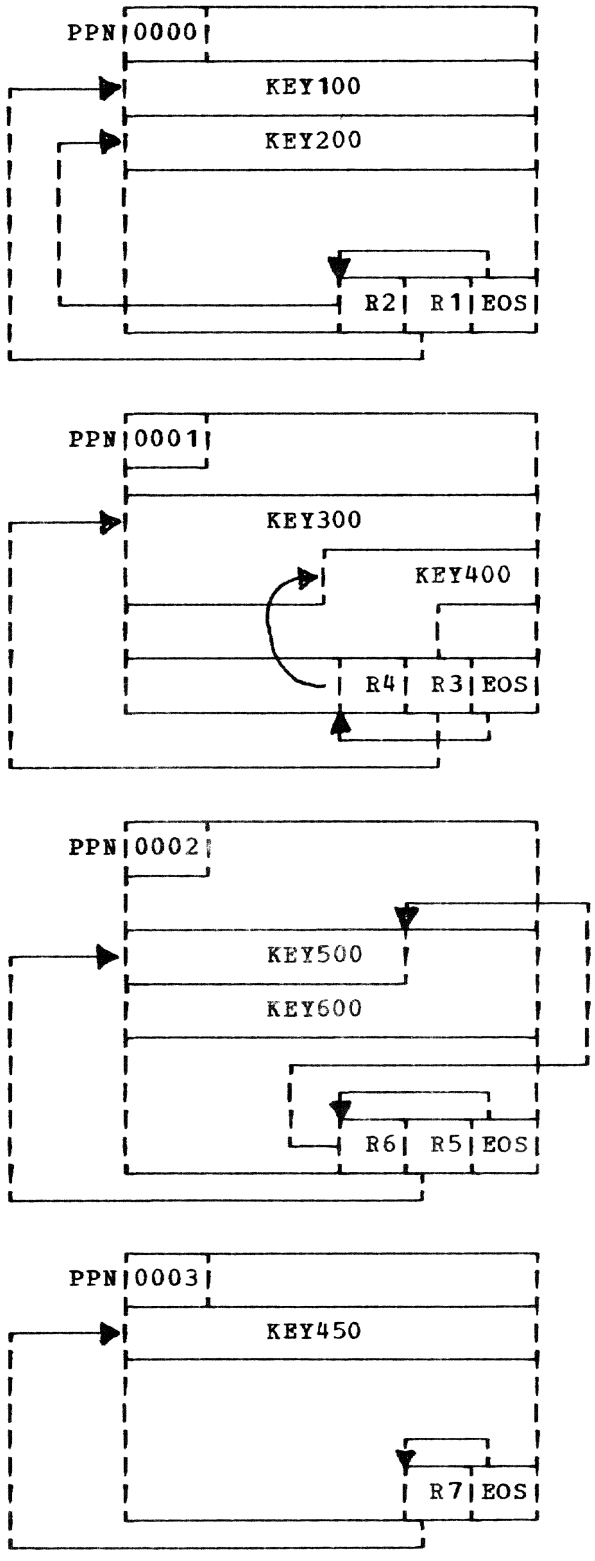
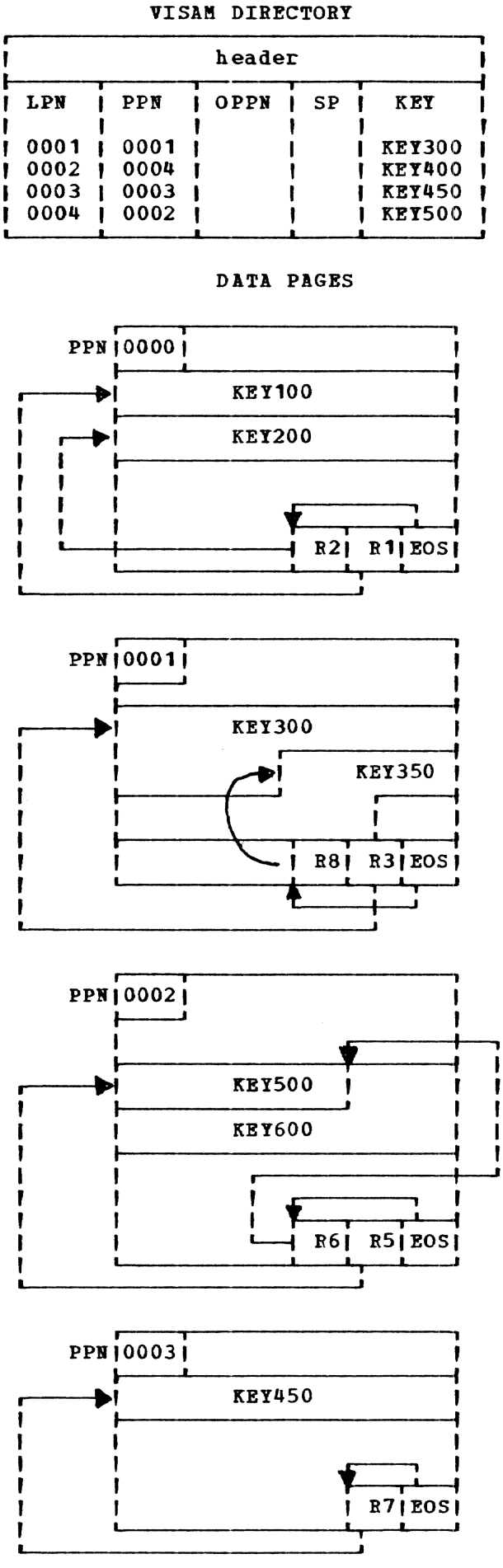


Figure 9B. Addition of record 7 KEY450 to Figure 9A



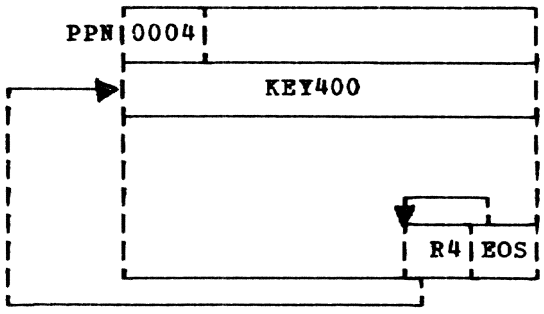


Figure 9C. Addition of record 8 KEY350 to Figure 9B

Optionally, the user can specify, in the DDEF command's DCB operand (or in the DCB macro instruction), that a certain percentage of space be left in each page during creation of the data set, for the addition of logical records (PAD parameter).

All buffering required for VISAM processing is supplied by the system. The buffer size is one page for data pages, one page for work page.

VISAM logical records may be format-F or format-V; detailed descriptions are in Appendix C.

The macro instructions associated with processing of virtual index sequential data sets are SETL, GET, PUT, READ, WRITE, and DELREC. For shared data sets, the ESETL and RELEX instructions are provided.

SETL positions a VIS data set to the beginning, end, next, previous record, or to any specified logical record within the data set. If the user wants to specify a particular logical record with SETL, he may do so by using either the record key or the retrieval address. However, for a shared data set, a user may not specify a retrieval address with SETL. As with the VSAM SETL, any attempt to position the data set outside its own bounds will cause an exit to the user's synchronous-error address (SYNAD). For a successful SETL, the record's retrieval address will be provided by the system in the appropriate DCB field.

GET obtains sequential access to a logical record of a VIS data set. It may be specified by the assembler user in one of two forms:

Move Mode -- The user provides the system with the address to which he wants the record transferred; the system moves it.

Locate Mode -- The user requests the address of the next logical record in the appropriate input buffer. With this address, he has the option of

processing the record in that location, or moving it to his own work area.

Again, after each execution of the GET macro instruction, the retrieval address of the logical record just retrieved is available in a data control block field.

PUT sequentially creates logical records in a VIS data set. They must be created sequentially. They must be presented to the system for concatenation with the data set in a logically ascending sequence of data keys. If a PUT macro instruction is issued for a record that has a key with a value that is less than or equal to that of the previous record, the system will detect this and exit to the user's synchronous-error routine.

This macro can be used when the DCB has been opened for output, if no other DCBs have been opened for output. It can be specified in either of two modes:

Move Mode -- The user provides the system with the address of a logical record, and the system transfers the record from that location to the next available output buffer segment; from there, it is automatically written to the output data set by the system before that portion of the buffer is released or reused.

Locate Mode -- The user requests from the system the address of the next available output buffer segment; he uses that address to store the logical record that he wants to add to the data set. The system automatically writes the record to the output data set when necessary.

As with VSAM, the VISAM PUT may be used as a means of truncating an already existing data set. If any records exist on this page beyond the current position, they are deleted one by one until the end of the page is reached. If any pages of this data set exist beyond this page, they are deleted. The directory is also truncated as necessary.

READ enables the user to read logical records nonsequentially, based on a user-supplied data key of the retrieval address. Since READ automatically uses SETL to position the data set at the proper record, it has the same limitiaion as SETL with regard to record specification; logical records of shared data sets may not be specified by retrieval address. After selecting a logical record from an index sequential data set or member, READ transfers that record to a user-specified location.

For shared VIS data sets, an exclusive-READ can also be specified by this macro instruction. Then no other program requesting that record can gain any access to it until it is released by the user who issued the READ macro instruction.

If an attempt is made to read a record with a key greater than the last key in the data set, the system transfers control to the user's end-of-data set address. If a READ request is made, and the record with the specified key cannot be found (but its key is less than the highest key in the data set), or if an invalid retrieval address is specified, control is transferred to the user's synchronous-error routine.

WRITE creates a VIS data set in a nonsequential manner, or inserts or updates logical records in an existing VS data set. The three basic functions of this instruction are:

WRITE -- New key

WRITE -- Replace by retrieval address

WRITE -- Replace by key operation

WRITE -- new key: The system assumes that the user wants to add a new record to the data set. A search is therefore made of the existing data keys in the data set; and exit is taken to the user's synchronous-error address, if a record with an identical key is found. If the key is unique, the system automatically positions the locator for the record in the appropriate position, so that the records of the data set will be available for retrieval in an ascending key sequence.

WRITE -- replace by retrieval address or WRITE -- replace-by-key: The system assumes that the user wants to update an existing record. If the system determines that the retrieval address or key specified is not that of an existing record, an exit is made to the user's synchronous-error routine. Otherwise, the system replaces the old record with the new one, adjusts the available space if the length of the new record is not equal to that of the old (placing the new record on an overflow page if necessary), and updates the record locators and maintains the logical key sequence.

For shared VIS data sets, the WRITE macro instruction also releases any page-level write interlocks placed on the record, through the same DCB, by an exclusive-READ.

DELREC deletes a specified record from a VIS data set. The user specifies, either by key or by retrieval address, the record to be deleted; DELREC uses SETL to

locate this record. If the record can not be found, an exit is made to the synchronous-error routine. If SETL locates the desired record, the locator for that record is removed from its page, the remaining locators are compressed, and the space occupied by that record is made available for future use. If the record with the lowest key on the page is deleted DELREC calls ADE (CZCPL) to update the directory to reflect the new low key on the page. When the last record on a data page is deleted DELREC will delete not only the record but its corresponding key entry from the directory and the page from the dataset. Page 0 is the only page which will not be deleted when it becomes empty. When a page is deleted from the dataset not only is its corresponding key entry removed from the directory but all key entries with PPN values greater than the page just deleted will be adjusted downward to reflect their new PPN in releationship to the dataset. The pages old PPN value is also saved in the key entry and is used when validity checking pages in the input page routine. (See Figure 9D.)

ESETL releases a page-level read interlock imposed by another macro instruction (e.g., GET, SETL or READ, nonexclusive) from a shared data set. It does not release the page-level write interlock set by an exclusive-READ.

RELEX makes a record that belongs to a shared data set available to other users, by releasing the page-level write interlock set by an exclusive-READ.

VISAM Sharing Rules: The use of VISAM with shared data sets results in setting and releasing interlocks.

DATA SET LEVEL INTERLOCKS -- If a VIS data set is opened for input, in-out, out-in, or update, a read interlock is set for the entire data set, preventing other users from opening it for output.

If a VIS data set is opened for output, a write interlock is set so that no other user can open it.

PAGE LEVEL INTERLOCKS -- A read interlock is set on a page of a VIS data set referred to by a SETL, GET or READ (nonexclusive) macro instruction; OPEN does not impose any page-level interlocks.

A page-level read interlock is released by an exclusive-READ, WRITE, ESETL, DELREC, or RELEX macro instruction, if issued against the data control block that caused the interlock to be set. Page-level read interlocks are also released when the data set is closed, or by any other macro instruction that refers to a page other than the current page. (for example, a sharer issues a READ macro instruction for a record,

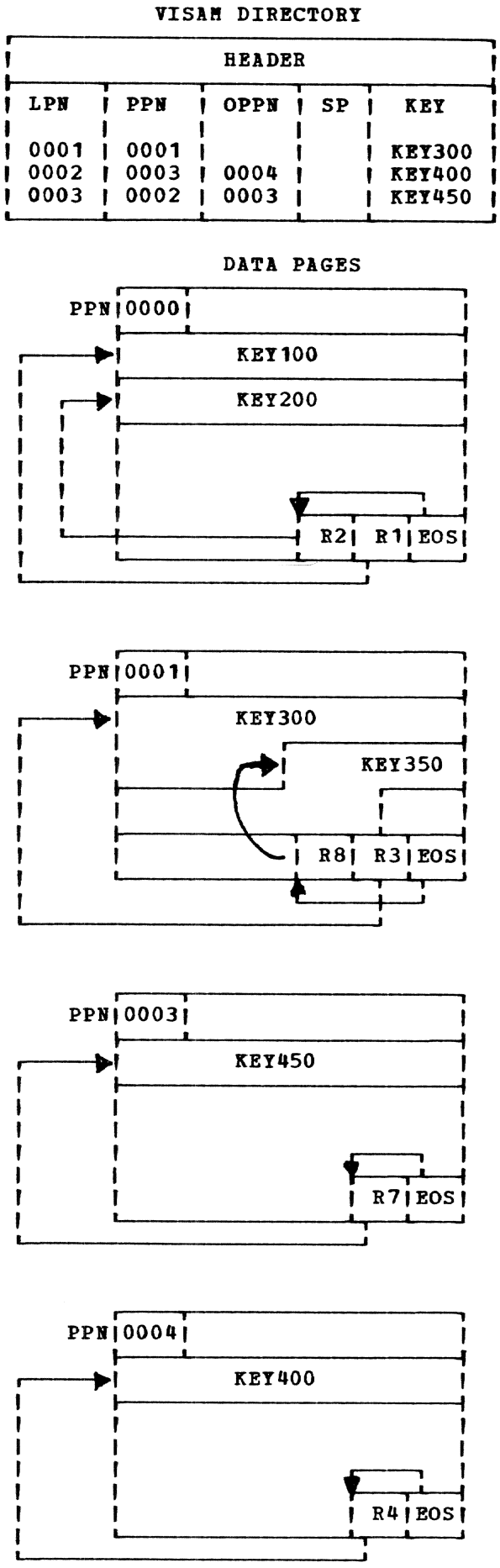


Figure 9D. Deletion of record 5 KEY500 and record 6 KEY600

causing a page to be brought into his virtual storage; later, he issues a READ for a record not on that page. The page-level read interlock, set when the first READ was issued, is released on execution of the second.)

A page-level write interlock is set by an exclusive-READ, or by a WRITE macro instruction.

A page-level write interlock is released by a GET, READ (nonexclusive), RELEX, WRITE, DELREC, or CLOSE macro instruction, or by any other macro instructions that refer to a page of the data set other than the current page.

Virtual Partitioned Access Method -- VPAM

The virtual partitioned access method is not an access method in the normal sense of the term. VPAM contains no routines for reading or writing records. A virtual partitioned data set really is a collection of data sets that a user has combined for ease of reference. These constituent data sets are called members; each member is organized as a virtual sequential or virtual indexed sequential data set. The other access methods are used to read records of a member into a task's virtual storage.

VPAM provides the control that performs these functions on members:

Create or add to a virtual partitioned data set.

Prepare any member of a virtual partitioned data set for processing.

Add new members to, or delete existing members from, an existing data set.

Update existing members in place.

Each member of a virtual partitioned data set is identified by the name of the virtual partitioned data set followed by an unqualified member name in parentheses. The partitioned organization (see Figure 10) allows the user to refer to either the entire data set or to any member of that data set.

References to individual members are made through the partitioned organization directory (POD). When a partitioned data set is created, a POD is set up to account for each member. As members are added, deleted, or changed, the directory information is automatically updated.

The first entry (one or more pages) in the virtual partitioned data set is the POD, which is used to locate members of the data set. Each member begins on a new page; any unused space on the preceding page is left open.

Provision is made for users to assign additional names, called aliases, to each member, and to locate each member on the basis of either its name or any of its aliases. The partitioned data set organization is suited for storage of libraries, where references to different entry points may require the loading of the same subroutine.

Example: A partitioned data set named MATHLIB, whose members consist of mathematical subroutines such as SQRT, ARCTAN, and COS, also contains an alias for SQRT, called ISQRT; this alias is used to indicate that the argument is a negative value, so an imaginary value is expected. References to both SQRT and ISQRT would indicate the same member, but a different entry point may be desired when ISQRT is named (see Figure 10).

Partitioned data sets may be composed of VS or VIS members, or a mixture of both.

All buffering required for VPAM processing is supplied by the system, based on the maximum logical length specified in the member's DCB; for a VIS member, the work areas needed for the ISD or POD are also supplied.

Two macro instructions are associated with VPAM: FIND is used to prepare a member for processing; STOW is used to update the POD and, in certain cases, disconnect a data set member from a user's problem program.

FIND searches a POD to locate the member descriptor of a particular VP data set member (using either the member name or any of its aliases), and then positions the member for processing. This positioning includes obtaining member information from the member descriptor and transmitting it to the member header in the RESTBL and to the DCB that has been opened for the data set.

FIND initially checks the DCB to determine if it is currently in use. If FIND had been issued previously for a member of that data set, and the information in the POD has not yet been updated by a STOW for that member, FIND calls STOW to update the member information in the POD. However, if the DCB is in use for the creation of a new member, that member may not yet have been named, so a STOW could not then be issued for it. Therefore, to protect against this situation, FIND will not attempt to issue STOW under these conditions, but will return an indication to the user that he must issue a STOW macro instruction for the new member before issuing FIND.

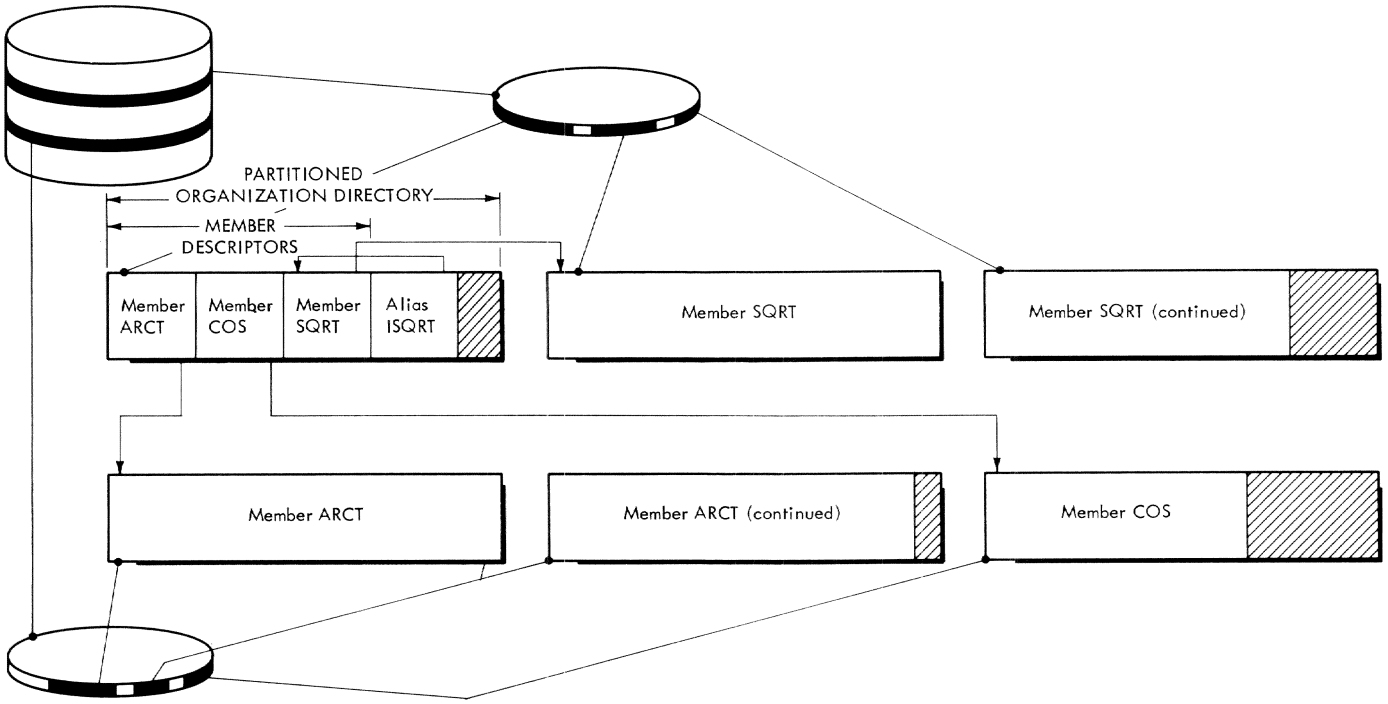


Figure 10. Virtual Partitioned Data Set

If the DCB is not in use, the POD is searched for the name given in the FIND macro instruction. If the name cannot be located in the POD, a not-found return is made to the user; if the name is located, sharing data is checked and the member is positioned for processing by the appropriate SETL.

FIND also provides the service option of moving user-data from the POD to a user-defined area.

STOW modifies, adds, or deletes member or alias descriptors in the POD; the processing will depend on the type of STOW specified by the user:

Type N (new) -- If the member name is not found in the POD, the POD is updated to reflect the addition of the new member. If the member name is found in the POD, processing is ended and a code returned to the user indicates that the new name is not unique.

Type NA (new alias) -- The POD is searched for each alias being added; if each is unique, alias descriptors are created.

Type R (replace) -- This type replaces user-data and closes the member. If "user area" is specified, the data will be stored in the POD. The POD is updated to reflect any changes made to the member, and return is made to the user.

Type U (update) -- Same as type R, except that the member header in the RESTBL is not closed; it remains active for further processing.

Type D (delete) -- This type causes the member to be deleted; all data pages associated with the member are deleted, and the member and alias descriptors are deleted from the POD. The DCB is initialized for reuse and control is returned to the user.

Type DA (delete alias) -- Deletes aliases from an existing member. The POD is searched for each alias being deleted and its descriptor is deleted from the POD. This process is repeated for each alias being deleted.

Types C and CA (change name and change alias) -- The POD is searched for the name or alias being changed. The new member name or alias replaces the old.

VPAM Processing: Since processing a VP data set usually involves only one member at a time, the single DCB opened for a data set can be used for the member being processed. For processing existing members, FIND must be issued after the OPEN macro instruction; however, when a new member is being added to the data set, a DCB is opened for either VIP or VSP (depending on the type of member desired), PUT or WRITE macro instructions are used to create the member, and a STOW (type N) is issued to include the member in the data set. In this case a FIND is not needed. A "FIND" is also not needed when the member name

parameter of the DDEF command is specified. For this case, OPENVAM will issue the "FIND". When several members are to be processed simultaneously, one DCB per member must be opened. The opening of each of these DCBs must be followed by a FIND macro instruction for that member, so that the appropriate information is placed in the correct DCB.

VPAM Sharing Rules: VP data sets are interlocked at the member level when a FIND macro instruction is issued; there are no interlocks set at the data set level, as for VSAM and VISAM. Member interlocks are set within the RESTBL when FIND is issued; they are released by the STOW or CLOSE macro instructions. Only the member being processed has the interlock applied; other members are available to other users for processing.

VIS members are:

write interlocked, when opened for output;

read interlocked, when opened with any other option.

VS members are:

read interlocked, when opened for input;

write interlocked, when opened with any other option.

SEQUENTIAL ACCESS METHODS

The sequential access methods directly specify the appropriate channel programs and they control the logic of error recovery, in addition to providing data set management. These access methods generally require that the user specify a large number of functions that are handled automatically by the virtual access methods. The user also has available to him special-purpose routines that enable him to create his own direct access and tape-volume labels. This is not possible with VAM.

Data sets accessed by the sequential access methods are of physical sequential organization. They are organized on the basis of physical records, whose order is determined strictly by the order of creation.

The sequential access methods are:

Basic sequential access method (BSAM)

Queued sequential access method (QSAM)

Multiple sequential access method (MSAM)

Terminal access method (TAMII)

Input/output request facility (IOREQ)

Basic Sequential Access Method -- BSAM

BSAM provides a limited data set compatibility with OS by supporting the direct access, or unlabeled, or standard labeled magnetic tape data set formats (except for the direct access split-cylinder format) that are produced by the OS basic sequential and queued sequential access methods. Also, BSAM is the primary means, within TSS, of accessing magnetic tapes.

BSAM creates the channel programs that sequentially access tapes or disks, and passes an I/O request control block (IORCB), containing the channel program and buffer information, to the resident supervisor through a supervisor call. The IORCB format is shown in Figure 11. The resident supervisor, in turn, initiates the channel program, records any pertinent error information, and passes the IORCB back to BSAM, which then attempts error recovery if necessary, and informs the user of the results of the I/O operation by posting the information in a data event control block (DECB). A DECB is a storage area reserved as part of a macro expansion (or reserved separately for future purposes by using the L-form) that relates an I/O operation to a specific READ or WRITE instruction. Each READ or WRITE requires one DECB that contains control information and pointers to status indicators.

With BSAM, the user must determine the outcome of his request before he can do any processing that is dependent on that request; the DECB provides a means for making the determination. The test for completion is made by issuing the CHECK macro instruction. If the I/O operation ends satisfactorily, control is given to the sequential instruction following the CHECK macro instruction. If the request results in an error or a special condition, control is passed to the user's synchronous-error routine (if one was specified; otherwise the task is terminated). If the I/O operation is not complete when CHECK is issued, the task will wait until the operation is complete.

BSAM creates its own channel programs in virtual storage, using virtual storage addresses. However, the channels do not operate on the basis of dynamic address translation, since they can not be made to wait for paging in whenever they reference a page that is not in main storage. For the same reason, all buffer areas that are to be referenced during the execution of a channel program must be in main storage during the entire I/O operation. Therefore, the resident supervisor reads the IORCB into its own area of main storage, translates the virtual addresses in the channel program into real addresses, and passes the IORCB back to virtual memory only when its buffer has been filled.

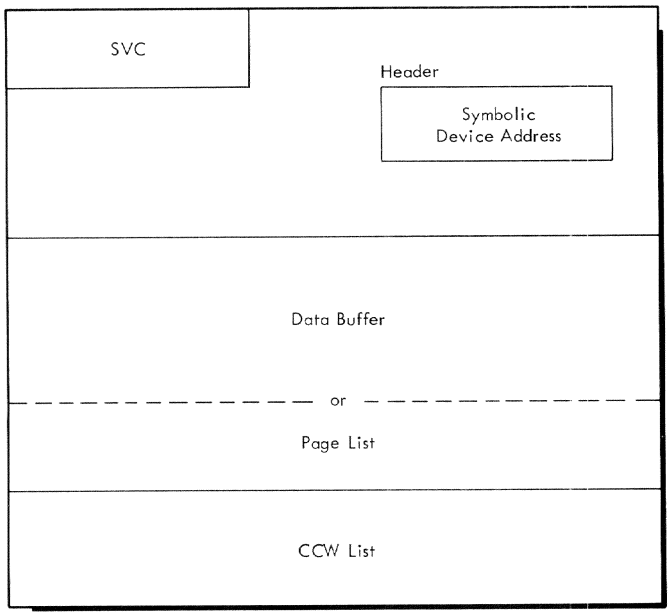


Figure 11. Input/Output Request Control Block (IORCB)

(Placing the IORCBs in supervisor storage serves another function: In general, BSAM buffers can be expected to be less than one page long. Since supervisor storage is allocated in 64-byte increments, the maximum size of an IORCB can be kept within 1920 bytes, thus saving paging overhead and main storage use.)

If a buffer is too large to be contained within the IORCB, BSAM places in the IORCB pointers to the pages containing the buffer.

Using BSAM: BSAM enables a user to access unblocked physical sequential data sets. It also provides access to blocked records; all blocking and unblocking must be done by the user. Whether records are blocked or unblocked, BSAM uses the block as the unit of data exchange with the problem program. BSAM accepts these record formats: format-F (blocked and unblocked), format-V (blocked and unblocked), and format-U (unblocked only). Descriptions of these formats are in Appendix C.

The system checks the physical lengths of blocks containing format-F records and transfers control to the user's SYNAD routine if an incorrect-length block is read. The user must then determine the size of the block read, from a count field in the DECB. Accordingly, the length of format-F records must not be changed after a data set is opened; the physical attributes of format-F records must be accurately described.

As with all access methods, before a user can employ BSAM to process a data set, he must open the DCB associated with that data set. In response to the BSAM OPEN macro instruction, the system:

Finds the matching data definition

Completes the DCB fields

Establishes address relationships and linkages to access routines

Issues to operator any required mounting messages

Verifies or creates data set labels

Positions volumes to the first record to be processed (see Table 1)

Allocates and prepares required buffer pools

Establishes the volume dispositions for end-of-volume conditions

Causes entries to user label checking, label creating, or DCB exit routines (if supplied).

In the CLOSE macro instruction, the magnetic-tape volume disposition is specified:

REREAD -- Reposition the current volume to reprocess its portion of the data set.

LEAVE -- Position the current volume to the end of its portion of the data set just processed.

For magnetic tape, the exact positioning that follows the CLOSE instruction will vary, depending on whether labels are specified for the data set. Table 2 defines two final-position numbers for labeled and unlabeled tapes. These numbers are then used in Table 3, which correlates the specifications of I/O processing in OPEN with the positioning specified in CLOSE.

BSAM Macro Instruction: These are in three general categories: data-set oriented, buffer oriented, and device-control oriented.

Buffering macro instructions -- BSAM is primarily intended for use on unbuffered physical sequential data sets; there is no automatic buffering facility. However, the user may provide himself with some buffering by using the GETBUF, GETPOOL, FREEBUF, and FREEPOOL macro instructions. All such buffers are only work areas for the user; they are not intermediate storage areas. All input/output operations between these areas and external storage are performed directly, without using intervening holding areas.

Table 1. Effect of OPEN Options

Open Option	Device	Action	Initial Positioning
INPUT	Magnetic-tape	Data set is read sequentially, either forward or backward (depending on what is specified in each READ macro instruction); labels, if specified, are processed as input	First data record.
	Direct access	Data set is read forward sequentially; labels, if specified, are processed as input	
RDBACK	Magnetic-tape	Data set is read sequentially, either forward or backward (depending on what is specified in each READ macro instruction); labels, if specified, are processed as input	Last data record of last volume of data set.
OUTPUT	Magnetic-tape or direct access	Data set is written sequentially; labels, if specified, are processed as output	If data set disposition is specified as NEW or OLD, volume is positioned to first data record; if data set disposition is specified as MOD, volume is positioned to one record beyond last data record of last volume of data set.
INOUT	Magnetic-tape or direct access	Data set is read sequentially first; labels, if specified, are normally treated as input (however, if records are written to the data set, subsequent labels, if specified, are processed as output); when reading is completed, volume is repositioned and data control block remains open so that data set can be processed as output	
OUTIN	Magnetic-tape or direct access	Initially data set labels are processed as output; after data set is opened, user may issue READ or WRITE macro instructions in any order; when end-of-volume is reached, labels are processed either as input or output, depending on whether READ or WRITE macro instruction caused end-of-volume condition	
UPDAT	Direct access	Data set is read sequentially; blocks can be updated in place by output requests that write last block read back to data set; labels, if specified, are processed as input	First data set record.

Table 2. Final Magnetic Tape Positions

	Labeled Tape	Unlabeled Tape
1	Preceding data set header label group	Preceding first data block of portion of data set resident on current volume
2	Following tape mark that terminates trailer-label group	Following tape mark that terminates last data block of portion of data set that is resident on current volume

GETPOOL requests allocation of a buffer pool area, and it assigns that area to a specific data control block. The user must specify the number of buffers in the pool, and their lengths. Only one buffer pool may be assigned to a data control block at one time.

GETBUF obtains a buffer from a specified buffer pool that must have been previously assigned to the data control block either by a GETPOOL macro instruction, or as a result of the buffer options specified in the DCB macro instruction. Buffers obtained by GETBUF must be returned by a FREEBUF, if they are to be obtained again.

FREEBUF returns (to its buffer pool) a buffer obtained by GETBUF. It is not necessary to free all buffers prior to closing a data set; it is necessary to free a buffer before it can be acquired again.

FREEPOOL releases areas that were previously assigned to specified data control blocks as buffer pools. The area must have been acquired either by the execution of a GETPOOL macro instruction, or as a result of buffer options specified in the DCB macro instruction. If a FREEPOOL has not been executed by the time a data set is closed, the CLOSE macro instruction will release the area involved.

Data set interactive macro instructions: READ, WRITE, CHECK, and DQDECB enable a user to:

Create a sequential data set by storing blocks in the order in which they were supplied.

Sequentially add blocks to the end of an existing sequential data set.

Sequentially retrieve blocks from an existing sequential data set, or retrieve an individual block based on these posiitioning capabilities -- beginning of data set, location of previous block processed by system, or location of any of data set's blocks.

Update an existing data set either by updating blocks in place, as sequential processing proceeds (direct access device only), or by updating blocks in a

Table 3. Effects of OPEN and CLOSE Options on Magnetic Tape Positioning

OPTION OF OPEN SPECIFIED AS	OTHER FACTORS INFLUENCING POSITIONING	DIRECTION OF LAST INPUT OPERATION	POSITIONING SPECIFIED IN CLOSE	
			LEAVE	REREAD
OUTPUT	None	Not applicable	Position 2	Position 1
OUTIN	None	Not applicable	Position 2	Position 1
INOUT	No WRITE operation executed for this data set	Backward	Position 1	Position 2
		Forward	Position 2	Position 1
	At least one WRITE operation for this data set	Not determining factor	Position 2	Position 1
INPUT	None	Backward	Position 1	Position 2
		Forward	Position 2	Position 1
RDBACK	None	Backward	Position 1	Position 2
		Forward	Position 2	Position 1

Note: Trailer label exits are taken for data set processed for INOUT or OUTIN, if last operation was a WRITE; no trailer label exits are taken if last operation was a READ.

nonsequential manner (direct access device only), or by reproducing a data set to allow the user to insert new records and/or delete old records as the modified copy is being made.

READ causes a request for a transfer of a physical record, from an I/O device directly to a specific virtual storage input area, to be recorded in a control block (DECB) and placed on an I/O request queue. Control is then returned to the user's program; when the device is available the request is executed.

WRITE causes a request for a transfer of a physical sequential record, from a specific storage area to an I/O device (directly, without using a buffer area), to be recorded in a control block (DECB) and placed on an I/O request queue. Control is then returned to the user's program; when the device is available, the request is executed.

CHECK checks the queue of control blocks (DECBs) containing the requests for read or write operations, to determine if those requests have been satisfied. It also indicates whether errors or exceptional conditions have occurred while satisfying the request. For each data set, the CHECK macro instructions must be issued in the same order in which the READ or WRITE operations were requested.

DQDECB removes all unchecked DECBs (created by issuing READ and WRITE macro instructions) from a queue of unchecked DECBs maintained by the system. This macro instruction is normally used in the SYNAD routine when multiple READ or WRITE macro instructions have been issued without an intervening CHECK. If DQDECB is issued, all unchecked READ or WRITE requests must be reissued. (The user must ensure, before reissuing, that the data set is positioned to the desired record.)

Device control macro instructions provide a user with physical control over a data set: BSP, CNTRL, FEOV, POINT, and NOTE. Some of these may be combined with the interactive macro instructions to provide nonsequential access to a data set, within the framework of BSAM.

BSP backspaces one physical record on the current magnetic tape or direct access volume. Regardless of the direction of reading (specified in the READ macro instruction), or the option specified in the OPEN macro instruction, backspacing is always toward the load-point on magnetic tape volumes or the corresponding position on direct access volumes.

CNTRL repositions magnetic-tape.

FEOV positions a multivolume data set at the beginning of the next sequential volume, before the physical end of the current volume is reached. This macro instruction is not applicable to data sets on unit record devices. When volumes are switched by this macro instruction, FEOV creates the necessary output labels for current and new volumes (output data sets) or verifies the volume labels for current and new volumes (input data sets). An attempt to execute this macro before all READ and WRITE requests to the data set have been checked will result in abnormal task termination.

POINT repositions a magnetic-tape volume to a specified physical record within a data set on that volume; for direct access volumes, POINT places control information in the appropriate control block, so that the indicated record will be the next accessed. The user must verify that the block identification previously provided by a NOTE macro instruction (now being used in the POINT macro instruction) refers to the same volume. Using POINT, in conjunction with the information provided by a previous NOTE, permits reading or writing a sequential data set from any specified position. All read or write requests must be checked for completion before the POINT macro instruction is executed.

NOTE makes available to the user the relative position within a volume of a physical record that has been just read or written. This relative position identifies the block for subsequent repositioning of the volume. Repositioning is normally accomplished by the POINT macro instruction. All read or write requests must be checked for completion before the NOTE macro instruction is executed.

Both the NOTE and POINT macro instructions require that the current block count in the DCB be valid. For an unlabeled data set, or a data set containing nonstandard labels, there are conditions when this count may not be valid, since the block count is normally found in the trailer label. These conditions occur when:

the DDEF command or macro instruction specifies a disposition parameter of MOD, or

the OPEN macro instruction specifies RDBACK.

Under these conditions, neither the NOTE nor POINT macro instructions should be used.

Practical Applications: A sequential data set can be created by using BSAM and specifying output or out-in in the OPEN macro instruction, and by using the WRITE and CHECK macro instructions to transfer blocks to the data set being created. To add blocks to an existing sequential data set, the user specifies output or out-in in the OPEN macro instruction, and MOD in the DDEF

command, to position the system to the end of the existing data set. He then issues a series of WRITE and CHECK macro instructions to add the physical records.

To obtain each of the physical records of a physical sequential data set in the order in which they were written, the user specifies input in the OPEN macro instruction to position to the first record of the data set. He then issues a series of READ and CHECK macro instructions to retrieve the blocks in sequence. It is also possible to retrieve the records of a physical sequential data set nonsequentially by using the NOTE and POINT macro instructions in the manner indicated in their descriptions.

Physical sequential data sets can be updated-in-place if they reside on direct access storage. When this method is applied, the user specifies updating in the OPEN macro instruction and then issues the appropriate sequence of macro instructions: READ and CHECK; WRITE and CHECK. Each READ and CHECK instruction provides a physical record in the user's work area. By examining this block (record), the program can decide if it is to be updated. If the record is not to be updated, the program can branch to another READ and CHECK instruction to examine the next block. If a block is to be updated, the program does that and then issues WRITE and CHECK macro instructions to return the just-read block, or its replacement, to the data set. (Only the most recently read block, or its replacement, may be updated and returned.) If two WRITE and CHECK macro instructions are issued without an intermediate READ and CHECK, the second WRITE overlays the first.

Queued Sequential Access Method

The queued sequential access method (QSAM) consists of the TSS data set management facilities that enable a user to access physical sequential data sets at the logical record level. QSAM, in contrast to BSAM, permits the user to store and retrieve logical records of a sequential data set without coding his own blocking/deblocking and buffering routines. Using QSAM, a sequential data set can be stored on, or retrieved from, disk or tape.

QSAM's basic functions are blocking and deblocking logical records, issuing I/O requests, and checking and positioning data blocks. QSAM itself blocks, deblocks, and buffers internally, but uses BSAM to perform I/O operations such as reading, writing, and checking and positioning for access to data.

Blocking Logical Records: QSAM blocks logical records according to the logical record-length and block-size parameters found in the DCB. When a user wants to include a logical record in an output data set, he issues a PUT macro instruction. QSAM adds this logical record to the physical record (block) currently being built if it will fit within the current buffer. If it will not fit, the block is considered complete, and the record for which the PUT was issued will be treated as the first record of a new block. The user can cause a block to be prematurely regarded as complete by issuing a TRUNC macro instruction.

Deblocking Logical Records: QSAM returns a single logical record to the user each time he issues a GET macro instruction. When the current block has been completely processed, the next GET instruction causes the buffer to be refilled, if the data set was opened for input or readback, or to be written back before refilling, if needed, when the data set was opened for updating. At any time, the user can cause processing of a buffer to be regarded as complete by issuing a RELSE macro instruction. Following this, the next GET macro instruction will retrieve the first logical record from the next physical record.

Buffering Blocks of Data: Double buffering is the normal buffering facility of QSAM. This involves the use of two buffers, one of which will be in use while I/O activity is being performed on the other. Thus, on a normal input or readback data set, while logical records from one buffer are being supplied to the user, the other buffer is being refilled. On a normal output data set, QSAM will continue adding logical records to one buffer while the other is being written out.

Under some circumstances, it is necessary to perform only single buffering; only one buffer is used. The decision to use double or single buffering is based on the OPEN option of the data set and on the macro option specified in the DCB. Double buffering will be done in all cases except when the data set is opened for updating, or SETL has been specified in the DCB.

Single buffering must be done on an update data set to allow the user to update one block of records at a time. No reading-ahead can be done until there is a determination on whether the current block of records must be updated, since an update WRITE instruction can return only the last block read. When the user specifies the SETL macro instruction, he must be able to specify it after QSAM finishes checking any individual physical I/O operation; single buffering is therefore a necessity.

Double buffering on a readback data set, with fixed or undefined length records, is handled in the same manner as for an input data set, except that blocks of records are read beginning with the last block of the data set. However, if a data set opened for readback specifies variable-length records, the procedure includes the use of a third buffer. After a block of records has been read and checked, a copy of it is

moved to the third buffer. This copy is used by the system as a table to contain record lengths, so that the records in the actual buffer may be accessed in reverse order. Note that, although three buffers are used, this is still only double buffering; the third buffer is, in a sense, a dummy.

Using QSAM: QSAM enables the user to access blocked and unblocked physical sequential data sets. The records within each such data set can be format-F (blocked or unblocked), format-V (blocked or unblocked), or format-U (unblocked only). These formats are described in Appendix C.

The OPEN macro instruction has the same basic functions in QSAM as the BSAM OPEN. In response to the CLOSE macro instruction, QSAM writes any remaining output buffers, disconnects the data set from the problem program, and takes care of any label writing and volume disposition that may have been specified. The effects of the OPEN and CLOSE options on magnetic-tape positioning are shown in Table 3. (Note: in-out and out-in are not supported in QSAM.)

As the user requests input or output of logical records, QSAM anticipates the need for I/O activity, manipulates buffers, and performs any deblocking or blocking that is required. The user is free to concentrate on processing of logical record streams, in and out of his program.

QSAM Macro Instructions: As with BSAM macro instructions, these are in three general categories: data-set oriented, buffer oriented, and device-control oriented.

Data-set oriented macro instructions enable a user to:

Create a sequential data set by sequentially storing its logical records in the order they are supplied by the user.

Sequentially add logical records at the end of an existing physical sequential data set.

Retrieve logical records from an existing physical sequential data set, or retrieve an individual record, based on these positioning capabilities:

beginning of data set on current volume,

end of data set on current volume,

previous logical record on volume (backspace),

or a record whose retrieval address was previously obtained.

Update an existing data set by updating logical records in place as sequential processing proceeds (direct access only).

The QSAM macro instructions are: SETL, GET, PUT, and PUTX.

SETL enables a user to logically position a physical sequential data set at its beginning, end, at the previous logical record, or at any user-specified logical record. Subsequent PUT or GET operations will start at the specified position.

GET reads logical records in sequential order; unless it is used in conjunction with SETL, when the order is not necessarily sequential. GET may be specified in either locate or move mode. In locate mode, GET locates the next sequential logical record of a data set, reads it into a buffer if necessary, and places its address in register 1. The user may then operate on the record in the buffer where it is located or he may move it to his own work area. In move mode, GET acquires the next sequential logical record from a buffer (reading it into the buffer if necessary), and moves it to a user-specified work area.

PUT writes new or altered logical records into a physical sequential output data set. PUT may be specified in either locate or move mode. In locate mode, PUT places in register 1 the address of an output buffer. The user should subsequently construct, at that address, the next logical record to be incorporated in an output data set. The system will automatically write the physical record, of which the logical record is a member, into the data set. In move mode, the PUT macro instruction moves a logical record from a user-specified work area into an output buffer, so that the system may include the record in the output data set. The user must ensure that the length of the logical record is in the proper DCB field before executing this macro instruction.

PUTX causes the next logical record in a buffer area of an input data set to be written as the next sequential logical record of an output or update data set. PUTX may be specified for either output or update mode. For update, the input and output data sets are one and the same; PUTX merely indicates to the system that a given logical record in a buffer associated with that data set is to be written back, in its present form, to the data set; for output, the input and output data sets are distinct; PUTX transfers a logical record from the buffer of the input data set to a buffer of the output data set, from which it is to be written out by the system. Note that PUTX (output mode) is effectively the same as PUT (move); in fact, the PUT macro instruction accomplishes this function more efficiently than PUTX. The

PUTX (output mode) instruction has been provided primarily as a conversion aid for OS users, since it provides a significant option under OS, in which exchange buffering is possible. For both update and output, the last macro instruction issued for the input data set, prior to PUTX, must be a locate-mode GET.

Buffer-oriented macro instructions, TRUNC and RELSE, give the user some control over system input and output for his data sets.

TRUNC causes the current output buffer to be regarded as filled, so the system will transfer the truncated physical record in that buffer, as it then stands, to the data set on the output device. The system is then positioned at the next buffer area, which will be used to hold the next logical record supplied, by the user, for output. If an attempt is made to execute this macro instruction when the output buffer is already full, or when the records are unblocked, the instruction will be ignored. Therefore, effective use of this macro always results in a nonstandard-length block being written to the data set.

RELSE causes the remaining records of the current input buffer to be ignored, locates the next sequential physical record's input buffer area, and positions the user at the first logical record in that buffer area. The next GET macro instruction will retrieve the first logical record from the new input buffer.

Device control-oriented macro instruction, FEOV.

FEOV directs the system to advance to the next volume of a data set before reaching the end of the current volume. It also ensures that the last buffer is written out to an output data set, and that any anticipatory requests to read, issued by the system for that volume but not yet checked, are purged. As in BSAM, when volumes are switched by this macro instruction FEOV creates the necessary output labels for current and new volumes (output data sets), or verifies the volume labels for the current and new volumes (input data sets).

Practical Applications: A physical sequential data set can be created, using QSAM, by specifying output in the OPEN macro instruction, and by using PUT macro instructions to transfer logical records to the data set being created. When the last record in the data set has been created, the user issues a CLOSE macro instruction. This writes the remaining output buffers, disconnects the data set from the problem program, and takes care of any label writing and volume disposition that may have been specified.

The user can add logical records to an existing physical sequential data set by specifying output in the OPEN macro instruction and modification (MOD) in the DDEF command; this positions the system to the end of the existing data set. He then issues a series of PUT macro instructions to supply the additional records. When all the additional records have been transferred, he issues a CLOSE macro instruction.

The logical records of a physical sequential data set may be retrieved in the order in which they were created. The user specifies input in the OPEN macro instruction to position the system to the first record of the data set, and then issues successive GET macro instructions to retrieve the logical records. When end-of-data is detected during a GET, the system transfers control to the user's end-of-data routine. Logical records may also be retrieved nonsequentially from a sequential data set by preceding the GET macro instruction with either the RELSE or the SETL macro instruction. The use of these macros has been previously explained.

The user may update physical sequential data sets in place, after specifying update in the OPEN macro instruction, by employing the PUTX macro instruction (update mode). First, he issues the GET (locate) macro instruction to determine the address of the next sequential logical record. By examining this record, the user can determine if he wants to update it. If it is not to be updated, a branch is made to another GET instruction, to examine the next record. If a record is to be updated, the appropriate changes can be made to it, and then a PUTX (update mode) macro instruction should be issued to return the updated logical record to its original storage location in the data set.

Multiple Sequential Access Method -- MSAM

MSAM consists of the data management facilities that enable the user to process logical records at the GET/PUT macro-instruction level for the IBM 2540 card reader/punch and the IBM 1403 printer. MSAM is a fast and efficient mechanism for simultaneously driving several unit-record devices under the control of a single task; MSAM also has automatic buffering and error-retry options.

MSAM differs from the other sequential access methods (such as BSAM). For each MSAM I/O request, the system processes a buffer group of physical records; for each BSAM I/O request, the system processes only one physical record. Considerable processing is required in the supervisor and the access methods for each I/O request, regardless of buffer size. Usually MSAM will make an I/O request only once to process each buffer, even though the buffer will contain a large number of physical

records; this is accomplished by chaining the channel command words (CCWs) related to each physical record in the buffer. System-processing overhead will thereby be minimized when using unit-record equipment.

MSAM also differs from the other sequential access methods because several data sets may be grouped on one device, allowing the user to process all of them under the same DCB. This saves him from issuing OPEN and CLOSE macro instructions for the DCB every time a data set with different characteristics is to be processed. Each data set is a data group. Input data groups may be separated by control cards, which MSAM will recognize and whose presence will be communicated to the user; he may then take whatever action is necessary. Output data groups on the card punch may be separated by the special cards that are automatically merged from the card reader, or the data groups may be physically removed from the stacker by issuing a message to the operator. The merging can be accomplished by specifying the COMBIN option in the DCB macro instruction; the removal, by issuing the FINISH macro instruction.

Each buffer used by MSAM (a buffer group of physical records) occupies one page of virtual storage. The number of buffer pages assigned to any DCB is based on the device with which the DCB is associated, determined individually by the specific installation by a parameter in the symbolic device allocation table (SDAT). This allows the value for a device to be adjusted so that the device will be driven full-speed for the maximum time between two consecutive time slices.

The first 32 bytes of each buffer page are reserved for control information used by MSAM. The remaining portion of the page is packed with logical records. The maximum number of such records per buffer page is 100 on input and 200 on output; depending on the size of the records, there may be fewer.

MSAM is well suited to the time-shared environment because it transfers responsibility for waiting for I/O completion from system service routines, such as BSAM check, to the invoking routine. Waiting for I/O while time-sharing is particularly undesirable during a user's time slice; a built-in wait-state is provided at time-slice-end. Therefore MSAM provides the facility for processing DCBs that are ready to be processed, and for skipping those that the user finds to require waiting. When all opened and accessed DCBs require waiting, the task may wait for the first I/O interruption associated with any DCB in the task.

Using MSAM: MSAM enables the user to access blocked and unblocked physical sequential data sets, when the data sets are associated with unit-record devices. Within each such data set, format-F and format-V records are permitted (see Appendix C).

The DCB defined for data sets that are to be accessed using MSAM includes a number of special fields (including the COMBIN field previously mentioned) that are not part of the DCBs generated for any other access method. When the user opens the DCB, the common portion of the OPEN routine completes the portion of the DCB that is common to all access methods, and then invokes the access-method-dependent OPEN routine. This routine allocates the required number of buffer pages, and allocates and formats an IORCB and a DECB for each buffer page that it allocates. The DECB is not generated at assembly time, as it is in other access methods.

When he has finished processing a data set with the MSAM macro instructions, the user issues the CLOSE macro instruction for that DCB. In response, the system returns all fields of the DCB to the conditions they were in before opening, issues the FINISH macro instruction (explained below), and releases the areas of storage obtained by the access-method-dependent portion of the OPEN macro instruction.

MSAM macro instructions are: SETUR, GET, PUT, and FINISH.

SETUR specifies the physical configuration of the unit-record device associated with the DCB for which this instruction is issued. When necessary, the system writes a message to the operator to notify him of the configuration he is to provide. Between repetitions of this macro instruction, the user must interrogate the DCBICB field of the DCB and, if it is non-0, invoke the interruption-inquiry routine by using the INTINQ macro instruction (described in Assembler User Macro Instructions) to determine whether an asynchronous interrupt is pending. If yes, the user must give control to the appropriate interruption-handling routine before reissuing SETUR.

GET obtains the next sequential logical record from an input buffer and may be specified in either the locate or move mode. In the locate mode, GET locates the next sequential record in the specified input data set, and places its address in register 1. In the move mode, GET locates the next sequential record in the specified input data set and moves it to a user-specified work area in virtual storage. The GET macro instruction of MSAM differs from GET in other access methods in the action taken when a referenced input buffer is not yet full. Instead of going into a wait state, MSAM returns a code to the user indicating that no record has been provided since the next sequential buffer has not yet been filled. To obtain that record, the

user must reissue the GET instruction; meanwhile, he may perform other work.

PUT includes a record in an output buffer, the contents of which are to be printed or punched on unit-record equipment. This macro instruction may be specified in either the locate mode or the move mode. When specified in the locate mode, PUT returns, in register 1, the address of an area within an output buffer. In this area, the user may construct a logical record which will automatically be included, by the system, as the next sequential record of the output data set. When specified in the move mode, PUT moves a logical record from a user-specified location to an output buffer; from there it will automatically be written as the next sequential record of the output data set. PUT returns to the user a code indicating the manner in which the instruction was completed. An I/O-not-complete indication informs the user that there was not enough room in a free buffer to include the logical record; he may reissue the PUT later, and, if a buffer is then free, the system will indicate by return code that the PUT was completed successfully. Again, it will automatically be written as the next sequential record of the output data set.

FINISH signals the MSAM routines that processing has been completed for the current data group (the current subsection of the data set). Employing this macro instruction, users can process data groups that have different attributes but are under the control of the same DCB, without closing and opening that DCB between data groups. FINISH initiates the final writing of buffers for an output data set, and tests the results of all outstanding I/O operations for both input and output data sets. To avoid having his task placed in a wait-state, the user should issue FINISH for a data set before issuing CLOSE. Rather than allowing the user to test for I/O completion, MSAM CLOSE will place the task in the wait-state until I/O activity is completed (MSAM CLOSE is the only MSAM routine that will do this). Another reason for issuing FINISH before CLOSE is to ensure notification of I/O errors on final I/O operations; CLOSE does not provide this facility. If the user receives a notification that I/O operations have not been completed, he may continue with other processing, and reissue FINISH at a later time. FINISH also will notify the operator to remove the current data group from the device; or it will automatically separate data groups being punched with cards from the card reader (under control of the COMBINE field of the DCB).

MSAM Error Processing: Provides the user with an automatic error-retry option, under the control of the DCB. Example: The DCB may specify that a print error be handled by striking out an erroneous line and attempting to print it again. The system will, if it is unable to recover from an I/O error encountered as a record is being processed, return an indication of this to the user; he can then determine whether the error was permanent. If permanent, the user should issue a CLOSE instruction for that DCB; if the error was not permanent, the user may continue processing records beyond the one with the error, by reissuing the macro instruction. For an input operation, he may even process the record with the error, since he will have a copy of it; however, the validity of that record will be doubtful.

Input/Output Request Facility

The input/output request facility (IOREQ) consists of the data management facilities that enable users to program their own I/O device-control routines. In effect, IOREQ is not an access method, but a means by which the user can create his own specialized access methods.

The user of IOREQ creates channel command words (CCWs) and executes them as he desires. Since the user of IOREQ can have complete control over a device, and possibly monopolize the channel to which the device is attached, the use of IOREQ is restricted to devices defined as private in the symbolic device allocation table (SDAT). Also, only the BULKIO task and E class users can request the allocation of a specific private device through a symbolic device address.

Because of the direct level of contact between this facility and the devices themselves, the user of IOREQ must:

Be thoroughly familiar with how the device interfaces with a channel through its control unit

Handle all exceptional conditions through his SYNAD routine

Reissue all outstanding requests if an I/O request is unsuccessful (perform his own error recovery)

Not exceed the maximum number of concurrent I/O requests for this device (specified in the SDAT).

The parameters for the channel program and buffer address, in an IORCB associated with each I/O request, must be explicitly defined by the user in IOREQ. While this places a greater burden on the user than in other access methods, it also provides him with greater flexibility. Example: He may specify a buffer located in an IORCB, or in a user work area; or he may write channel programs that use CCW chaining and he may perform scatter-reads or gather-writes (reading or writing data into or from vir-

tual storage locations that are not contiguous).

Another feature of IOREQ is that channel programs may be command-chained in the channel. When the channel completes the channel program in one IORCB, if command chaining was specified, the channel immediately begins executing the program established in a second IORCB that has been made available. With this option, IOREQ users who are reading or writing large amounts of data (too large to fit in a single IORCB) can employ buffering, by linking the IORCBs.

Using IOREQ: As with the other access methods, the user must open a data set before using IOREQ to access it; a CLOSE macro instruction must be issued to disconnect the data set from the system.

In response to the OPEN macro instruction, the normal open-common functions are performed first. Then the access-method-dependent portion of the open routine is given control; tests ensure that the user is privileged to access the specific volume and device, that IOREQ has been specified as the DDEF operand, and that the device to be used is defined as private in the SDAT. Storage is allocated for the data extent block (DEB) and the IORCBs; information is moved from the JFCB to the DEB.

When a user wants to execute one or a series of I/O operations, he issues the IOREQ macro instruction. At assembly time, this instruction generates a DECB that will be used to store the completion status of the operation. This control block is interrogated by the CHECK macro instruction to determine when and how the operation has been completed. The operation, or series of operations, are explicitly defined by the user in the VCCW macro instruction.

When the CLOSE instruction is issued, the task is put into the wait-state until all outstanding I/O requests have been completed; then all storage allocated during open-processing is freed, and the normal close-procedures are completed.

IOREQ Macro Instructions: VCCW, IOREQ, and CHECK.

VCCW generates the virtual channel command word, a doubleword that contains the information that will prepare the IOREQ macro instruction for the requested I/O activity. Through the use of chaining fields, groups of these doublewords, generated in successive storage locations, can be made to form VCCW lists. The user can specify read, write, or read-back operations, as well as no operation (NOP), sense, and transfer in channel (TIC); or, he can specify a hexadecimal command code.

IOREQ initiates a sequence of I/O operations that are specified in the previously generated list of virtual channel command words. IOREQ uses this list as input for generating a list of channel command words (CCWs) to be placed in the IORCB for execution by the appropriate channel. IORCBs are executed separately by the channel, unless the user specifies IORCB chaining in the appropriate field of the VCCW. (Note: This is not the same as the VCCW chaining accomplished within the IORCB.) IORCB chaining is allowed only between IORCBs that are on the same device. Even though IORCBs may be chained, separate CHECK macro instructions must be issued for each IOREQ result, because each IOREQ generates a separate DECB.

If buffering is specified for an IOREQ, the size of the buffer within the IORCB for read-request VCCWs is determined by the difference between the lowest and highest data-area addresses specified in any read-request VCCW within that VCCW list; some data areas may overlap. Therefore, the user must ensure that a contiguous entity is formed by the individual data areas referenced by each read-request VCCW in the list associated with that IOREQ.

The size of the buffer built for write-request VCCWs is determined by the sum of the individual data areas associated with each VCCW; that is, unique buffer space is allocated for each write-request VCCW, regardless of whether the data areas referenced by these VCCWs have overlapping portions. Consequently, the data areas associated with write-request VCCWs do not need to form contiguous areas.

When buffering is specified in IOREQ, data is moved from user data areas to output buffers within the IORCB before any I/O activity is performed for any of the write-request VCCWs within the VCCW list. Therefore, although a user may chain VCCWs that are to read into a particular data area and then write from that area, the sequence of operations will result in the old, not the new, data being written, as the user might expect.

If buffering is not specified in IOREQ, the area within the IORCB that would normally have been used for the buffers is used instead for page-list entries to the user's data areas. Then the data transfer is directly between the channel and these areas.

CHECK tests for completion of an IOREQ macro instruction, and detects errors and exceptional conditions. CHECK must be used for every IOREQ issued; and must be issued in the same order as the IOREQs. If an exceptional condition is detected, control is passed to the user's SYNAD

routine (which must be provided or an ABEND will be executed). If the I/O operation is successful, the user's program resumes execution at the instruction following the CHECK instruction.

TERMINAL ACCESS METHOD -- TAMII

The Terminal Access Method (TAMII) handles all TSS communications. This includes communicating with local and remote terminal users, SYSIN and SYSOUT datasets and local and remote systems or RJE work stations. TAMII is used by both the system and the user by issuing GATE and T-GATE macros (GATRD or TGATRD etc.). The T-GATE macros are extensions of the GATE macros and allow the I/O to be overlapped.

TAMII is composed of five distinct components. They are:

1. RTAM - Real Terminal Access Method

   This component resides in the supervisor and is an interface between the TSS supervisor and the device modules (DCMs; see below)

2. DCM - Device Control Modules

   The DCMs provide all device dependent support required to do the following functions:

   a. builds channel programs and initiates I/O

   b. maintains line control during non-activity between user and task

   c. handles device dependent timer routines

   d. validates task I/O requests

   e. handles device dependent PCI requests and non-normal completion status

   f. handles the connection of a device to a task whether initiated by the user or the task

   g. sets up device dependent information in the required system control blocks

   h. provides error recovery for all abnormal endings

   i. checks user's input for user function requests (cancel, attention, etc.)

   j. determines length and type of input

   k. provides simple output edit capability for system messages to the terminal user.

3. VTSS - Virtual Terminal Support System

   This component resides in the task's virtual memory, validates the requests, and translates the program's request against the user's environment. After determining what has to be done, VTSS will call the correct format control module (FCM) to format the data if any for the user's terminal.

4. FCM - Format Control Modules

   The FCM performs the required editing and/or formatting required by both the terminal device and the user. The FCM also ensures that the request is setup in a mode that the next level of TAMII will understand. The FCM then calls the module or access method, either in real core or virtual memory, required to do the actual request. The FCM provides the following functions:

   a. edits output data

   b. translates output data to line code

   c. invokes correct routine to do I/O

   d. translates input data to EBCDIC from line code

   e. edits input data

   f. moves input data to correct data area (user's or GATE's)

   g. sets up correct return code

   h. performs any requested valid control functions

   i. maintains correct sequence and buffer links for buffered requests in virtual memory

   j. performs any special task initialization required for connecting device

5. TCS - Terminal Command Subsystem

   TCS handles all device command requests and maintains the terminal environment control blocks.

Unlike other acccess methods, TAMII does not use DCBs and JFCBs; the primary control blocks are a TCT (Terminal Control Table) entry for RTAM and the FCL (Format Control Library) entry for VTSS. At the time the terminal is connected to the system, a TCT

is allocated and constructed. When the terminal is connected to a task a FCL is allocated and constructed by VTSS. Both the TCT and the FCL contain device information and areas to be used for work areas. For communication between VTSS and RTAM the ATCS SVC, with an associated parameter list, is used to initiate or request a function to be performed at the terminal. When RTAM communicates with VTSS, it is through a special I/O synchronous interrupt processor.

TAMII also will use other access methods to fulfill the program's request. Since TAMII is the access method used to read and write SYSIN and SYSOUT non-conversationally TAMII has to be able to access datasets. TAMII does this by using the appropiate access method for the dataset. TAMII supports VAM and QSAM and can read and write datasets with DSORGs of PS, VI and VS.

TAMII allows both queued and direct control of the terminals by the application program. The application program may also transmit EBCDIC character strings, in which case TAMII handles all editing and translation, or the program can bypass the TAMII character facilities and transmit direct terminal control information.

Through appropiate default values, TAMII allows the task owner to control the queuing function independently for both input and output, transparent to the application. This allows the application programmer greater control over the testing of his application program.

TAMII assumes all responsibility for error recovery. When the application issues a request, TAMII assumes the responsibility of getting the request to the terminal. For output, under normal conditions, the application does not receive a completion notification if the write completes successfully.

Using TAMII

TAMII is used both by the system and by the application programmer when either issues the appropiate GATE or T-GATE macros. Currently the application programmer is unable to initiate the connection of a terminal to the task. Therefore there is no 'OPEN' macro as such. The terminal user has to initiate the connection by entering a 'BEGIN' command at the terminal and then the application is informed of the connection request. The following macros are available to the application programmer and the system for controling the terminals:

1. CHCKT - check a DECB for completion of a request.
2. DIAL - dial a terminal through an autocall mechanism.
3. EXLIST - activate, deactivate terminal exit list entries.
4. FINDQ - poll and locate work for an application.
5. SETTERM - allows user to set, reset and interrogate flags and fields in the TAMII control blocks.
6. SOLICIT - solicit data input from a terminal by using an increasing number prompt or a decrement count of lines.
7. TCLEAR - purge active and pending I/O requests for a terminal.
8. TCNTRL - initiate a control request for a terminal.
9. TDCMD - execute a string of device control commands for a terminal.
10. TFREE - release a terminal from the task and the system.
11. TGATRD - read an input line from the pending input queue or the terminal.
12. TGATWS - write a message to the user's primary SYSOUT.
13. TGATWR - write data to a terminal
14. TGTWAR - write data and read any available input from a terminal.
15. TGTWSR - write a message to the primary SYSOUT and read the user's response from the primary SYSIN.
16. TRCBUF - read a line from the conversational buffer for the terminal.
17. TWRTLST - write a list of output to the terminal.
18. TERMPRO - set up or save a terminal user's environment.
19. TRANLCD - locate a translate table for a terminal.

Three macros which allow an application to push and pop a terminal's environment and pending queues after an attention from the terminal user are:

1. ATTNSAV - save the current terminal environment and pending queues.
2. ATTNRST - restore a previously saved terminal environment and pending I/O queue.
3. ATTNDST - destroy a previously saved entry.

For application programs, supporting multiple terminals and/or users, four macros are available to setup the terminal

controls and control the terminal connections:

1. MTT - inform and setup control blocks connecting terminals to an application when the request is initiated by the terminal user.

2. MTTDCN - disconnect and discontinue the multiple terminal application program.

3. ILOGON - inhibit user initiated connections.

4. PLOGON - permit user-initiated connections.

TAMII provides five ways for an application program to be informed of the completion of a request or of the receipt of an asynchronous interrupt; they are:

1. Device 'EXIT LIST' - when the exit list condition occurs, the routine identified by the exit list is scheduled to receive control. This list is device specific.

2. Application program general 'EXIT LIST' - a list general to the program. When a device condition occurs which the device exit list does not have an entry for, the application program's general list is checked and if it has an entry, the routine is scheduled for execution.

3. The system SIR and DIR with SIEC and SAEC (synchronous I/O and asynchronous I/O interrupt) queuing mechanism is supported by TAMII. If there isn't an exit list entry, the FINDQ work table is marked and an interrupt is queued by calling the Task Monitor.

Note: items 1 through 3 above all result in an asynchronous call to the application's routines.

4. FINDQ work polling capability - the application may process completions synchronously to execution by issuing the FINDQ macro when the program is ready to receive interrupt and completion notification.

5. CHCKT - TAMII 'check' capability, using the techinque of assigning a DECB to a request and later issuing the TAMII CHCKT macro for the DECB to determine if the request has completed. With the TAMII CHCKT function an application program can specify whether a wait is to be done if the I/O has not completed.

All TAMII macros are keyworded and most of them use the same keywords and return the same return codes to help the user in coding. When supporting multiple terminals, the most important keyword is the 'USN'. This keyword is the application's way of telling TAMII which terminal the program is communicating with.

TAMII has a common return code set for its macros. This helps the programmer in using the TAMII macros.

Code	Explanation
0	successful call
4	terminal is busy (request queued)
8	attention received on this request
12	request aborted attention pending
16	request purged by TCLEAR request
20	end of data received for SOLICIT
24	user error in parameter list
28	input not available for TGATRD
32	requested operation not supported on this terminal
36	terminal disconnected from system
40	permanent error on request

PART III: USE OF DATA MANAGEMENT FACILITIES

The user of TSS may have no direct use for many of the data management facilities. Interfaces are provided to request for him specific data management routines that will perform specific services.

Although assembler users normally have the most direct contact with data management facilities because they employ the macro instructions of the access methods, usually they cannot directly access terminals using RTAM; some use the GATE macro instructions as interface when they need the RTAM facilities. The MTT-mode macro instructions (see Part II, under "RTAM") provide this interface for multiterminal tasks.

All users can employ the command system to create, access, and modify data sets; the command system, in turn, requests the facilities of the appropriate access method.

I/O routines of the FORTRAN and PL/I (F) libraries provide the interface between the compiled code and the system's data management routines for FORTRAN and PL/I (F) users.

ASSEMBLER INTERFACES

The nonprivileged assembler user has no direct communication with either unit-record equipment or terminals from within his problem program. However, he can indirectly access unit-record equipment, and his own terminal, by means of the bulk-output facilities and the GATE macro instructions. Bulk-output facilities are much the same as those in the command language. See "Command System Interfaces," next in this part.

The GATE macro instructions allow the nonprivileged assembler user, from within his problem program, to write to his own SYSOUT, to read from his own SYSIN, or both. Depending on whether the task in which they reside is conversational or nonconversational, the GATE routines call on TAMII or VAM to accomplish their functions.

The GATE routines process any required writing by dividing the message into device-sized lines, or smaller; then the appropriate access method is determined and used to transmit the message to SYSOUT.

When reading is required, the GATE routines determine the appropriate access method and use it to obtain the input message; they apply a predefined character-translation table to the message as it is transmitted to the user's buffer.

The GATE macro instructions: GATRD, GATWR, GTWAR, GTWRC, and GTWSR.

GATRD reads a record from a SYSIN device, translates it to internal code, and places it in a user-designated area of virtual storage.

GATWR translates a record that is stored in a user-defined area, and writes it on a SYSOUT device.

GTWAR translates a record that is stored in a user-defined area and writes it on a SYSOUT device; then it reads a record from the SYSIN device and places that in another user-defined area of virtual storage.

GTWRC processes in the same manner as GATWR, except for nonconversational SYSOUT records, in which it translates a record and a carriage-control character that is stored in a user-defined area, and then passes it to a SYSOUT device.

GTWSR (for conversational tasks only) translates a record that is stored in a user-defined area, and writes it on a SYSOUT device; then reads a record from the terminal and places it in another user-defined area of virtual storage.

MCAST is an assembler macro instruction that allows the user to replace the character-translation table with one of his own choosing; this new table will be used by the GATE macro instructions for the duration of the task, to translate data transferred between the user's program and SYSIN or SYSOUT.

The command system uses the basic data management facilities to get a broad range of data management services; the user can enter, manipulate, output, and copy data sets; he can enter and delete data set catalog entries, and he can utilize the catalog-sharing facilities described in Part II.

There are five categories of command system data-management services:

Text-editor services
DATA-command services
Data-set copying services
Bulk input/output services
Data-set cataloging services

Details on the interfaces that will be outlined here can be found in the Command System User's Guide.

TEXT EDITOR

With the text editor, the user can create or alter a virtual index sequential (VIS) data set. It interfaces with the GATE and VISAM routines to perform the requested data management services.

The VIS data sets created and operated upon by the text editor are either region or line data sets. A region data set is indexed by a key consisting of two fields, a region name and a line number; region names, arranged alphabetically, divide the data set into regions; line numbers index the elements of each region. The line number is a seven-digit decimal number at the beginning of each record.

A line data set is indexed solely by line number; although it can be thought of as a special class of region data set (with a null region name), line and region data sets have different maximum record lengths (see Appendix C for record formats).

The text editor commands: EDIT, CONTEXT, CORRECT, EXCERPT, EXCISE, INSERT, LIST, LOCATE, NUMBER, REGION, REVISE, and UPDATE.

EDIT invokes the services of the text editor. If the user has not previously defined, by issuing a DDEF for it, the data set named in this command, a text editor routine will automatically issue a DDEF with a standard set of operands. In this case, the DDNAME issued for this data set will be EDDNnnnn, where nnnn is a number that is automatically incremented within a task for each new DDNAME issued, to preserve uniqueness.

If the data set to be edited exists, or has had a DDEF issued for it, it must be a VIS data set, or a VIS member of a virtual partitioned data set. The user will be prompted if either of these conditions has not been met. Also, he will be prompted if the data set is read-only, since it is assumed that he wishes to alter it.

After a JFCB has been created or located for the data set, the DCB associated with the data set will be opened; the DCB is located within a module of the text editor. If the data set is partitioned, the user's entry of a member name is verified. If none was entered, the user is prompted and an exit is taken; if a member name has been entered, a FIND macro instruction is issued for that member. If the member is new, an entry is made in the POD by the STOW macro instruction; if the member exists, a check is made to ensure that it is virtual index sequential. After all initialization, return is made to the command mode for further text editor commands.

CONTEXT replaces a specified character string with an input character string, wherever it occurs within a given range of lines. After checking the input for validity, CONTEXT issues a VISAM SETL for positioning at the first line with in the specified range; then it issues a GET for that line (record). The line is checked for occurrences of the specified string, which is replaced if found. After the line has been completely searched, it is written out by a VISAM WRITE, if any replacements were made. SETL is then issued for any necessary repositioning, and GET is issued for the next record. This process is repeated until the range of lines has been completely checked.

CORRECT makes corrections to a line or a range of lines within an object data set. If only one line is to be corrected, the CORRECT routine uses GATWR to print that line, before correction, on the user's SYSOUT. The VISAM SETL and GET macro instructions are used to obtain that line and subsequent lines from the object data set. Then the SYSIN macro instruction is used to obtain the user's corrections from his SYSIN. A new line is constructed in an output buffer, based on

these corrections, and the VISAM WRITE (replace-by-key) is used to write the line back to the object data set.

EXCERPT incorporates a portion of a line or region data set into the line or region data set currently being edited. On entry to this routine, the data set to be sampled is opened. Abnormalities in opening (e.g., data set not found, or not VIS data set or data set member) result in user prompting; an error-exit will be taken. If the data set was opened successfully, VISAM SETLs and GETs will be used to obtain the records to be incorporated, starting with the specified (or defaulted) first line. The lines will then be renumbered (that is, their keys will be changed); using WRITE (new key), the resultant lines will be written out to the data set being edited. If the REVISE command had been specified previously, indicating that the lines being excerpted are replacing existing lines, the previously existing lines already will have been deleted by REVISE, so there will be no key conflict with WRITE (new key). (See also REVISE command.)

EXCISE deletes a line or a range of lines from a line or region data set. VISAM SETL and GET are used to position to the desired line within the data set being edited; then DELREC is used to delete the record by key.

INSERT prepares the text editor to accept data lines for insertion following a given line in the source data set. The SYSIN macro instruction obtains the input data from the user's SYSIN.

LIST places a line or a range of lines on a user's SYSOUT. Lines of a region or line data set are retrieved with a VISAM GET macro instruction and listed on the user's SYSOUT with a GATWR macro instruction.

LOCATE searches a specified range of lines in an object data set for a specified character string. VISAM SETL and GET retrieve the lines within the range sequentially, until the specified string is found within a line; then GATWR prints the line with that string on the user's SYSOUT.

NUMBER renumbers a range of lines within a region or line data set; in effect, this associates a new key with each record within the range. NUMBER uses VISAM GET (locate mode) to obtain the lines within the specified range; as they are obtained, they are placed in a deletion list, to be deleted (by key) by the DELREC macro instruction. The keys are then changed to conform with the specified renumbering; the changed records are placed in an addition list from which they will be placed in the object data set by a VISAM WRITE (new key).

REGION prefixes a region name to a line number or range of line numbers; the lines so prefixed form a region data set and their keys consist of the combination of the region name and line number. Since the region name is a part of the record key, and seven characters of the key are reserved for the line number, the key length specified in the DCB for the data set being edited must be greater than 7, to allow room for the region name, which will be truncated to fit if necessary. (The key length parameter is computed and inserted as part of the EDIT processing; it is computed as the sum of 7 plus the value of the REGSIZE parameter in the user's profile.) The user will receive an error message if he attempts to provide a region name within a data set whose key length is not greater than 7. The SETL macro instruction positions to the next available line in the specified region; for a new region, this will be line 100.

REVISE prepares the text editor to accept data for inclusion, at a given point, in the object data set. It accomplishes this by first deleting all existing lines within the specified range, using the VISAM DELREC macro instruction, and then positioning the data set at the beginning of the range; the user can then enter replacement lines. The user will be prompted if an attempt is made to enter more lines than the range allows.

UPDATE prepares the text editor to accept new or replacement data lines, from the user's SYSIN, that are to be placed in his object data set. The SYSIN macro instruction is used to read the data; if the records are not variable length, they are padded with blanks as needed. UPDATE then checks the key supplied by the user at the beginning of the record (if no key was provided, the user will be prompted). If a line with that key exists in the data set, the record is written to the data set with a VISAM WRITE (replace-by-key); otherwise, a VISAM WRITE (new key) is used.

SERVICES OF THE DATA COMMAND

The command system's data-editing services allow the user to build and edit both VS and VIS data sets. The DATA, MODIFY, and LINE? commands are in this group. MODIFY and LINE? are used only with VIS data sets, and interface primarily with VISAM routines; DATA is used for VIS and VS

data sets, and interfaces with both VSAM and VISAM.

DATA creates either a VIS line data set or a VS data set; also it allows the user, during the creation of a line data set, to dynamically insert, delete, and replace lines in that data set.

After validating its input parameters, DATA verifies that a JFCB exists for the named data set (that is, if the user has issued a DDEF for that data set). If it exists, the JFCB must show either that the data set is a virtual partitioned data set, or that it has VS or VIS organization; if either of these conditions is not met, an error message will be issued to the user. If a JFCB does not exist for that data set, DATA will create one by issuing a DDEF; in this case, the data set name and organization are as specified in the input parameters, and the data definition name is derived from a value maintained by the system for this purpose.

DATA now opens the data set and, if it is partitioned, issues a FIND macro instruction to ensure that the member name is unique. All further processing depends on the type of data set being created, VS or VIS.

For a VS data set, the SYSIN macro instruction prompts the user with a number sign and retrieves the record from the user's SYSIN. Input records continue to be read until one is found containing either a %E, or an underscore followed by a command; either of these signal the end of input.

For a VIS data set, the user is prompted for input with the current line number. Another difference is that DATA must check input records for modification indicators; if DATA finds a %D followed by a line number, the line indicated is deleted from the data set being built by the DELREC macro instruction. If a line number preceded by only a % is found, the text following the % is written either as a replacement or as an insertion line, depending on whether the line number specified exists in the data set. If it exists, a VISAM WRITE (replace-by-key) is issued; if the line number does not exist, a VISAM WRITE (new key) is issued. As for a VS data set, end-of-input is indicated either by an underscore followed by a command, or by a record containing a %E.

When an end-of-input indicator is reached, DATA closes the opened data set (a STOW instruction is issued for the member, if it is a virtual partitioned data set) and then passes control to the proper routine.

If an attention interruption is received while the DATA command is in operation, further processing depends on whether the data set has been opened. If it has not been opened, DATA merely returns control, leaving the JFCB set up if one was generated. If the data set has been opened, DATA closes it (issuing a STOW instruction for the member if it is a VP data set) and then returns control.

MODIFY inserts, deletes, replaces, and reviews records in a VISAM data set or VISAM member of a VP data set. Also it may be used to build a new VISAM data set or member. In contrast to the data sets operated upon by the DATA command, MODIFY may be used with VISAM data sets that are not line data sets. This is possible since the user may specify, as part of the MODIFY parameters, an arbitrary key length and displacement, as well as record-format indicators.

After input parameters have been validated, MODIFY searches for a JFCB for the named data set. If none exists (that is, no DDEF has been issued for the data set), a JFCB will be created for it. In either case, the JFCB must show that the data set is either a VP or VIS data set, and that the user may write on it.

When the JFCB is located or created, the data set is opened. If the data set is partitioned, a FIND macro instruction must be issued for the specified member name. If the name is found, a check is made to ensure that the data set has VIS organization; if it is not found, a new member is created with this name and with VIS organization.

Input records containing the user's modifications are obtained, one at a time, by the SYSIN macro instruction. The user-supplied key points to the location of the specified record. When the user-input does not indicate deletion or revision, the record is written into the data set as an insertion or replacement, using either WRITE (replace-by-key) or WRITE (new record). When the first character supplied by the user is D, the record at the specified location is deleted from the data set by the DELREC macro instruction; when the first character is R, the record is reviewed (presented to the user), by the GATWR macro instruction. If review of all modifications is requested, the record that is being replaced or deleted will be presented to the user before the modification is made; for insertions, the

reccord immediately preceding the insertion is presented.

When the end-of-input record is reached, the data set is closed; for a VP data set, the STOW macro instruction is issued to reflect any alterations before the data set is closed.

LINE? presents to a user's SYSOUT the contents of specified lines from a line data set, or a language processor list data set. (A list data set is similar to a line data set; each record in a list data set has a unique line number. However, unlike the line data set, these records must be fixed-length and the line numbers are at the ends of the records.)

After checking and validating the data set name presented by the user, LINE? initiates a search for a JFCB bearing the indicated data set name. If no JFCB is found, DDEF is called to create one. Then the data set is opened and, if it is partitioned, a FIND macro instruction is issued to locate the indicated member. If it is a list data set, a check is made to verify that it is in list format; otherwise a check is made for line format.

The VISAM SETL macro instruction positions the data set at the beginning of the range specified by the user. Successive GET macro instructions obtain the records; a check is made to ensure that the specified range has not been exceeded. The GATWR macro instruction is then used to write the record on the user's SYSOUT (if the object is a list data set, the line number is written separately before the record so it will precede the record on the user's SYSOUT).

After all indicated lines have been processed, the data set is closed (if it is a VP data set, a STOW instruction is issued for the member), and control is returned.

DATA SET COPYING SERVICES

The command system provides facilities for making additional copies of existing data sets; depending on the particular command he selects, the user may also be able to change the medium on which the data set exists. The data set copying commands: VT, TV, VV, and CDS.

VT copies a data set that is in one of the VAM organizations (VS, VIS, or VP) to magnetic tape, as a physical sequential data set. There is no simple correspondence between the records of the VAM data set and the records of the physical sequential data set. Records of the physical sequential data set created by this command are blocked into page-length segments, regardless of the record sizes in the original data set. Therefore, it would be futile to attempt to use one of the sequential access methods (for example, BSAM) to obtain the records as originally placed in the virtual storage data set. The user can employ the TV command (described below) to copy the data set back to a direct access device, at a later time, and then access it with one of the virtual access methods.

Initially, VT checks that the input data set is a VAM data set. If there is no JFCB for the indicated input data set name, one will be created. For the output data set, the user must have created a JFCB (with a DDEF) that has a data definition name (DDNAME) of DDVTOUT. VT locates this JFCB and verifies that it indicates the proper data set organization (physical sequential) and the proper device type.

When the data sets are opened, the JFCB of the input data set and the common portion of the input data set's format-E DSCB are written as the first record on the output tape. The remainder of this record is padded with 0's.

Data pages to be copied from the input data set are located through its RESTBL. For each of these pages, the system's paging mechanism is used for input; each page is then written to tape by BSAM WRITE. Eight buffers are used to overlap processing time and input/output time.

After the tape operation has been completed, both data sets are closed and all buffers are released. Unless specified otherwise, the output data set is cataloged and any JFCBs created by VT are released.

TV retrieves and writes into a virtual storage volume, a data set previously written on magnetic tape by a VT command. TV verifies that the input data set has physical sequential organization and that it resides on a tape volume. If a JFCB for the input data set cannot be found, a DDEF is issued to create one. A check is then made that the output data set name indicates a new data set, with a virtual storage organization. If an output JFCB is not located, one is created by issuing a DDEF. The user must issue a DDEF for the output data set only if he wants it to reside on a private volume.

After the data sets have been opened, the first record is read from the input set (see "VT," above, for content), to verify the tape format, and make availa-

ble the DSCB data necessary to recreate the original data set. Data records from the tape are input by BSAM READ and output (to the direct access device) by VSAM PUT. At this time, the data set is being treated as VS, with page-length records, regardless of how it was originally created by the user. For these operations, eight buffers are used. The initial instructions to read tape fill the eight buffers; subsequently, four buffers at a time are filled as the other four are emptied, to overlap processing time and input/output time.

After all instructions for the input/output operations have been issued, both data sets are closed and all buffers are released. If the output data set, on direct access storage, is not completed correctly, the command-system ERASE routine is called to delete the partial data set. If the output data set has been completed correctly, the DSCB now associated with this data set must be modified, since it now reflects the organization of the data set as it was created by VSAM PUT (with page-size records and VS organization). To correct this DSCB, the format-E DSCB that is part of the output data set is read in by TV, and the DSCB for the newly-created data set is updated from the information in the old one. The DSCB then reflects the structure of the data set as it was originally created by the user. A catalog entry for the output data set is created if required; any JFCBs created by TV are released.

VV makes a copy of an existing VAM data set; the copy will also exist as a VS data set. Initially, VV verifies that the input data set name is the name of a VAM data set. If the user does not issue a DDEF for the specified data set name, there will not be a JFCB for it. A JFCB will be created by a call to DDEF (a catalog entry, to act as an input source, must exist for the data set if a JFCB is to be created). The output data set name must indicate a new VAM data set. Again, if no JFCB exists for this output data set, one is created by a call to DDEF. The user needs to create a JFCB for the output data set only if he wants it to reside on a private volume.

When both data sets have been opened, the common portion of the input data set's format-E DSCB is retained for recreating the data set structure after the copy operation has been completed. Data pages to be copied from the input data set are located by indexing through the RESTBL; the system's paging facilities read these pages. They are then written to the output data set with a VSAM PUT; at this point the output data set is being treated as a VS data set, with page-size blocks.

When the copy is complete, both data sets are closed and all buffers are released. If the output data set has not been completed correctly, ERASE is called to delete the partial data set. For normally completed VV operations, the output data set's DSCBs reflect the structure of the data set as it was created by VSAM PUT (VS structure, page-size records). Therefore, the DSCB is updated from the DSCB information retained from the original data set, so that structure of the data set is shown as it was created by the user. The catalog is updated, if necessary, and any JFCBs created by VV are released.

CDS copies a data set, or member of a VP data set; it may also copy members of a partitioned data set (with user data and aliases) into a second VP data set, replacing or ignoring duplicate members. CDS provides the user with the option of specifying that the original data set (or member) be erased after duplication; he may also renumber a line data set while copying it.

Initially, operands are checked for validity, and a JFCB is obtained or created for the input and output data sets. If the input and output data sets are both VP and no member name has been specified, multiple member processing (copying members with user data and aliases, if they exist) is assumed.

For multiple member processing, three DCBs are opened; one for input, one for VSAM output, and one for VISAM output. If no member names have been specified for the input data set, then every member found in the input data set's POD will be copied. Otherwise, only the members specified will be copied. A FIND for a member is done, which fills in the input DCB and obtains the user data for the member. The output POD is searched to see if a member with the same name exists. Then each alias in the input POD which is associated with the member is checked in the output POD. If a duplicate alias is found, it must be associated with the same member name in the output POD or processing of the member is ended. If no invalid duplicate aliases are found, and the user has not specified that duplicate members are to be ignored, the input member is copied into the output data set using the appropriate output DCB . When the copy is complete, the input member is erased if applicable, and the output member is added to the output POD with its user data and aliases, using STOW. Multiple member processing is com-

plete when all specified members have been copied.

If multiple members are not being processed, the input DCB is opened and checked against the output JFCB. Both data sets must have the same organization (VAM or physical sequential). Any combination of VAM data sets may be copied. If a VS data set is being copied to VIS, the keylength, relative key position, and pad must be specified for the output data set (since these may not be obtained from the input). In all other combinations, the output is given the same DCB parameters as the input. For VS format-U records, a LRECL of one page is used. The output DCB is then opened.

For physical sequential data sets, SAM READs and WRITEs are used to obtain the input records and place them in the output data set. For VAM data sets, VAM GETs and PUTs are used. If renumbering is specified, the input record is obtained and the new key is overlaid on the old before the record is written. Normal processing ends when the input data set is exhausted.

When processing is complete, the DCBs are closed and the input data set is erased, if specified (not applicable to multiple member processing). Control is then returned to the calling routine.

BULK INPUT/OUTPUT SERVICES

Because of the suitability of public storage for the operating environment of TSS, users may often want to transfer data sets that are on cards or tape volumes to public VAM volumes. Alternatively, some may want to write data sets to tape, punch them on cards, or print them on the installation's high-speed printer. Some of these functions can be accomplished by using the data set copying services. Other options, notably those involving unit-record devices, are performed using the command system's bulk input-output services. These services consist of the PRINT, PUNCH, RT, or WT commands, together with the operator-assisted card input facility. The user can issue only the commands associated with the output of data sets (PRINT, PUNCH, and WT). The system operator must initiate the others.

Bulk Output: When the user issues a PRINT or PUNCH command, the action taken depends on the nature of the data set to be printed or punched. Private data sets will be handled by the PRINT and PUNCH routines outlined below, and a separate nonconversational task will be created for this purpose. Public data set printing and punching will be handled by the BULKIO task. This will not be described here. The WT command routine, described below, is used for both public and private data set writing. VSAM and VISAM are used for operations on VS and VIS data sets, respectively; BSAM is used to control tape I/O, and MSAM is used to access unit-record devices.

PRINT will print an existing private physical sequential, virtual sequential, or virtual index sequential data set, on an installation's on-line high-speed printer. If a physical sequential data set is being used for input, it must be on a tape volume. Since a physical sequential data set can be allotted to only one task at a time, and the nonconversational task created by PRINT will require it for input, specifying a physical sequential (PS) data set to be printed will result in the release of any JFCB for the data set within the task which issued the PRINT. Also, if a PS data set is not cataloged, it will be automatically cataloged when PRINT is issued; it will be erased when the nonconversational PRINT task is completed.

On initial entry to the PRINT routine, this command determines the devices to be used and the input data set organization; it issues DDEFs for the input and output data sets, opens these data sets, and obtains any buffers that will be needed. The MSAM SETUR macro instruction is issued so that the printer has the required device configuration.

After setting up an identifying output line, PRINT obtains input records by an internal buffering technique (using VSAM or VISAM GETs, and BSAM WRITEs and CHECKs), and writes them to the printer with internal buffering (using MSAM PUTs and INTINQs). PRINT continues to loop in this manner, until the last record has been printed; then, it indicates any records that were received in error on the task's SYSOUT. The input and output data sets are closed, and the nonconversational task is finally logged off.

PUNCH is used to punch a cataloged VS or VI private data set into cards on an installation's high-speed punch. When the nonconversational task created by this command receives control, it calls DDEF to define the input and output data sets; it then opens each of these data sets, and issues an MSAM SETUR for the punch (output data set), to ensure that the proper card form is mounted. One logical record at a time is then read, by VSAM or VISAM GETs; after each record is read, control options are tested, and the record is written to the output buffer with MSAM PUT. This reading and writing continues

until all input records have been processed; then, input and output data sets are closed, and final messages are written to SYSOUT, including a count of the number of records read, punched, and skipped, and the number of error records. An exit is then taken, and LOGOFF called.

WT writes an existing VS or VIS data set on tape, for eventual printing on a high-speed printer. The output data set is automatically formatted into print lines, the format required for high-speed printing. After the operations necessary to log on the nonconversational task, WT calls DDEF to define the input and output data sets, and opens these data sets and output buffers with a BSAM GETBUF macro instruction. A blank line is constructed to provide for initial page positioning. The first record is obtained with a VSAM or VISAM GET. An internal buffering routine writes the records to the output data set, using the BSAM WRITE and CHECK macro instructions. After all input records have been read and written onto tape, the output buffers are released, the input and output data sets are closed, and, if requested, the output data set is cataloged. WT then writes on SYSOUT the number of records read, written, and skipped, and the number of error records. Then LOGOFF is called to terminate the task.

Bulk Input: If the user has data sets on tape or cards that he wants to have read into the system as bulk input, he must submit them together with any required information to the system operator, who will then enter and catalog them under the user's ID (userid).

RT is issued by the system operator, on behalf of a user, to read a physical sequential data set from tape, convert it to a VAM organization (VS or VIS), write it to a public VAM volume, and catalog it in the user's catalog. VIS data sets will be built as line data sets.

When the system is ready to process this command, it creates the necessary nonconversational task, and requests the operator to mount the input tape. After calling DDEF to create the JFCBs for the input and output data sets, RT opens both data sets and obtains input buffers with the GETBUF macro instruction. Input records are read by an internal buffering routine, which uses the BSAM READ and CHECK instructions. The records are then written out with the VSAM or VISAM PUT; if they are line data sets, the record lengths and line numbers are inserted.

When all input data has been read, the input and output data sets are closed, the output data set is cataloged (in the user's catalog), and the input buffers are released with the BSAM FREEBUF macro instruction. Then the record counts and an end-of-task message are written to SYSOUT, and LOGOFF is called to terminate the nonconversational task.

OPERATOR-ASSISTED CARD INPUT

The user can submit his data sets on punched cards to the system operator, who will then enter them into the system through the installation's high-speed card reader. Two types of input are permitted: non-conversational SYSIN data sets, and data-card data sets. The two types may be interspersed in any order within a batch of cards. No command is necessary to read the cards into the system; control over the reading of these card data sets is part of the function of the bulk I/O task.

A SYSIN data set contains all commands needed to run a nonconversational task. When one of these data sets is read in, it becomes the SYSIN data set of a nonconversational task, with the submitter's user ID. It will be executed as soon as space is available. After execution, the SYSIN data set is eliminated from the catalog and system storage.

A data-card data set contains any information the user wants read into public storage as a cataloged data set. As it is read, a virtual storage data set is created in public storage and cataloged in the catalog of the user who submitted the data set. It will reside in storage until it is specifically erased. (For details of the formats of both card data sets, see Command System User's Guide.

DATA SET CATALOGING SERVICES

The command system gives the user the facility to explicitly request that a catalog entry by created, altered, or deleted. The commands for these purposes are: CATALOG, DELETE, and EVV. The ERASE command, in addition to freeing all direct access storage assigned to a specified data set, deletes from the user's catalog the entry associated with a data set.

CATALOG creates or alters catalog entries for specific data sets. The user can also create a data-set superstructure, called a generation data group (GDG), to exercise catalog control over future structural elements (generations).

This command can be used to change a data set name in an existing catalog entry for both VAM and physical sequen-

tial data sets (except ASCII tapes). However, with this command, a new catalog entry can be created only for physical sequential data sets. A catalog entry can be created, with the EVV command, for a VAM data set for which no entry exists (either because one was deleted, or because the data set was created at another installation). When a data set resides on a direct access device and its name is being changed, the DSCB on the volume for that data set is updated to reflect the change.

When a GDG is being created, the user must initially issue the CATALOG command, naming the new GDG, to set up an index entry in the catalog; he also indicates at this time such control information as the number of generations to be retained as part of the GDG. Other data sets can then be cataloged as new generations of the GDG.

DELETE removes a data set entry from a user's catalog. An entry for any private data set can be removed with this command. The original catalog entry for a public data set, however, cannot be deleted; this is a protection against the system "losing" the data set (unlike a private data set, the JFCB for a public data set does not contain enough information to locate the data set). The sharer of a public data set may delete the entry in his catalog; however, the data set owner's entry is not affected (the sharer's entry consists only of pointers to the owner's catalog entry). If the owner of a public data set attempts to delete his catalog entry, he will receive a diagnostic message; no action will be taken.

EVV catalogs all the VAM data sets on a private VAM volume. The system's paging facilities are used to read in the DSCBs associated with each data set on the volume; as each is read in, the data set associated with it is cataloged, based on the information in the DSCB. When processing is completed, the private device that was required for mounting the private volumes is released.

FORTRAN users of TSS have little direct contact with the system's data management facilities. They must issue a DDEF for any data sets (except SYSIN and SYSOUT) that they expect their program to access, and they must specify in the DDEF a data definition name (of the form FTxxFyyy). Beyond that, the FORTRAN library modules provide the major data management interface; within these, the I/O control module (CHCIC) is the primary point of interaction. (A description of how the FORTRAN user of TSS specifies data set characteristics is in FORTRAN Programmer's Guide, GC28-2025.)

FORTRAN I/O CONTROL

The FORTRAN I/O control routine fulfills I/O requests made through other I/O library routines by using the data management macro instruction facilities of TSS. The data management facilities to be used are determined by the type of I/O statement issued in the user's program and by related DDEF commands that define such things as the type of records being transferred, and the manner in which they should be processed.

In general, either VAM or BSAM macro instructions may be used. When BSAM is used, the control routine employs its own internal buffering to speed up processing. The list of FORTRAN statements, below, identifies the principal macro instructions used for each statement; of course, other instructions, such as OPEN and CLOSE, must be used in conjunction with these.

READ obtains a logical record from a user-specified input source by using the READ, GATRD, or GET macro instruction.

WRITE initializes writing a logical record by establishing pointers to the output buffer area. Subsequent output processing is performed by using the WRITE, GATWR, or PUT macro instruction.

REWIND repositions the user-specified volume of one or more data sets to the first record of the first data set by using the POINT or SETL macro instruction.

BACKSPACE repositions the user-specified data set to the previous logical record by using the NOTE, POINT, SETL, and BSP macro instructions.

ENDFILE defines the end of the user-specified data set by using the WRITE and STOW macro instructions.

PL/I (F) INTERFACES

Like FORTRAN, PL/I (F) provides library modules that greatly simplify the use of the system's data management routines. The user need only issue DDEFs describing each data set, other than SYSIN or SYSOUT, that he expects his programs to access, and follow PL/I (F) language requirements for specifying data characteristics. (PL/I (F) Programmer's Guide and PL/I (F) Language Reference Manual tell how to specify data characteristics from within the PL/I (F) language.)

For the DISPLAY statement, library module IHEWDSP is a direct interface between the compiled code and the GATE macro facilities. For STREAM I/O, there is no single interface with compiled code; the type of STREAM I/O statement being executed determines which library module is invoked by compiled code. Each STREAM I/O statement finally invokes module IHEWIOF to issue the macro instruction. For RECORD I/O, the single interface with compiled code is module IHEWION, which interprets the I/O request, verifies its validity, and calls the library module that issues the appropriate macro instruction.

Table 4 summarizes the PL/I (F) interface with data management.

Table 4. PL/I (F) Interface With Data Management

TYPE OF I/O	FILE DECLARATION	ACCESS METHOD USED	MODULE ISSUING MACRO INSTRUCTION	PL/I I/O STATEMENT	MACRO INSTRUCTION
DISPLAY		TAMII	IHEWDSP	DISPLAY	GATWR and/or GATRD
STREAM		QSAM, VSAM, or TAMII	IHEWIOF	GET	GET (move mode) or SYSIN
				PUT	PUT (move mode) or GATWR
RECORD	CONSECUTIVE, SEQUENTIAL, BUFFERED	QSAM or VSAM	IHEWITG	READ	GET (locate mode)
				WRITE	PUT (locate mode)
				LOCATE	PUT (locate mode)
				REWRITE	PUTX
	CONSECUTIVE, SEQUENTIAL, UNBUFFERED	BSAM	IHEWITB	READ	READ
				WRITE	WRITE
				REWRITE	WRITE
	INDEXED, SEQUENTIAL, BUFFERED or UNBUFFERED	VISAM	IHEWITD (for format-F) IHEWITN (for format-V)	READ	ESETL or SETL to position, GET (locate mode) to read
				WRITE	PUT (locate mode)
				LOCATE	PUT (locate mode)
				REWRITE	WRITE
				DELETE	DELREC
	INDEXED, DIRECT, BUFFERED or UNBUFFERED	VISAM	IHEWITE (for format-F) IHEWITM (for format-V)	READ	READ
				WRITE	WRITE
				REWRITE	WRITE
				DELETE	DELREC

Time Sharing uses several groups of labels to identify direct access and magnetic tape volumes, and the data sets they contain, on secondary storage. The labels, used to locate the data sets, are identified and verified by the label processing routines.

The use of standard labels enables the system to identify volumes and ensure that the correct volume is being used and that no current information is inadvertently destroyed.

DIRECT ACCESS VOLUMES

Direct access volumes play a major role in TSS; they are used to store

- Privileged service routines
- The system catalog
- The system library

In addition, all public storage resides on direct access volumes. All data sets in public storage are organized as virtual access method (VAM) data sets. A direct access volume used for private data sets may contain all VAM type or all physical sequential type data sets but not both types on one volume.

VAM Data Sets

With the exception of tracks 0 and 1 on cylinder 1 (which are reserved for system-generated volume information), each VAM-formatted direct access volume is arranged into a succession of contiguous pages, each 4096 bytes long. The first accessible page of the volume (which starts on byte 1, track 2, cylinder 1) is referred to as relative page 0, and all other pages are numbered consecutively. Other pages need not begin on track boundaries. Locations in a volume are referenced by relative page number rather than by cylinder and track numbers.

The standard volume label, resident on cylinder 0, track 1, contains a pointer to the page assignment table (PAT) which is one page long for volumes of type 2311, two pages for type 2314, six pages for type 3330, twelve pages for type 333B, and sixteen pages for type 3350. The page assignment table contains a one-byte field for each page in the volume, arranged in sequence, and is used for the allocation of the pages on the volume. This field indicates whether the page is free and available for writing, in current use as a data-set page, in use as a DSCB page, or unavailable.

A data set page contains part of a data set. A data set control block (DSCB) of Format-E is associated with each data set. Each DSCB is 256 bytes in extent with a 44-byte key containing the data set name. DSCBs reside on DSCB pages, 16 DSCBs to a page. They are not necessarily on the same volume as the data sets (or individual pages of the data sets) to which they refer. The 212-byte data portion of the DSCB contains the description of the data set and its location in storage, by volume and page numbers. If the DSCB is not long enough to contain the list of all page numbers, the additional information is contained in one or more type-F DSCBs.

When a data set is created, space is allocated for its data information and for the associated DSCB. A data set descriptor (DSD) is placed in the user's catalog entry and gives the location of the format-E DSCB which in turn gives the location of the data set's data pages.

VAM data sets on public storage are always mounted, but VAM-formatted private volumes must be mounted before the data can be accessed. Accordingly, the DSD for a data set on a private volume must also contain a pointer to the volume on which the data set resides, if it resides on one volume, and to the volume on which the E-format DSCB resides, if the data set extends over more than one volume.

The DSCBs contain pointers to the public volume table (PVT) which is maintained by the system in shared virtual storage. Volumes are referenced by relative volume numbers which the public volume table translates into symbolic volume addresses.

Standard Volume Label (Figure 12): The standard volume label resides on cylinder 0, track 1 of the volume. The fields in the label are the same as those in the magnetic tape volume label, described in table 4, with these exceptions:

PAT page pointer	Contains the relative page number of the beginning of the PAT.

Device type — Contains the device type: 2311, 2314, 3330, 333B, or 3350.

Volume status indicator — Indicates the status of the volume: '00' = private volume, '20' = public volume

Public volume number — Contains the relative volume number of this volume within a set of public volumes.

VAM format indicator — Indicates VAM format: 'V2' = VAM2 format

Note: System programmers using the Time Sharing Support System (TSSS) must expect an 84 byte volume label; the label, as described in Figure 12, will be prefixed by a key of characters "VOL1" and the fields will be displaced by 4 bytes.

Format-E DSCB (Figure 13): This format is common to all VAM data sets. The format-E DSCB is the data set label; it corresponds to a tape header label. Also, it describes up to 38 pages of the data set. The format of the external page entries is described in Figure 14.

Format-F DSCB (Figure 15): This format is used to describe additional pages of a VAM data set, if there are more than can be described in the format-E DSCB. If additional space is needed, this DSCB will point to another format-F DSCB. The format of the external page entries is described in Figure 14.

VOL (label identifier)	1 (volume label number)	Volume serial number	Volume security	PAT page pointer	Device type	Volume status indicator	Public volume number	Reserved (currently blank)	Owner name-and-address code	VAM Format indicator	Reserved (currently blank)
1-3	4	5-10	11	12-13	14-17	18	19-20	21-41	42-51	52-53	54-80

Figure 12. Standard Volume Label (VAM only)

Data set name	System code	Pad for index sequential	No. of bytes used in last data page	Record format	Spare	Data set organization	(12 bits unused) Record length (20 bits)	Spare	Key length	Key location	Secondary allocation	No. of data pages	No. of directory pages	No. of overflow pages	No. of private vols
1-44	45-57	58	59-60	61	62	63-64	65-68	69	70	71-72	73-76	77-78	79-80	81	82

Total no. of pages assigned (Data set size at CLOSE)	Spare	Reference date	Change date	Spare	List of volumes for private data set (each 6 bytes, variable length) / External page entries (for public vols) (each 4 bytes long)	External page entries, each 4 bytes long (on full-word boundaries)	Pointer to next DSCB if chained	DSCB type	DSCB ID	Check sum
83-84	85	86-88	89-91	92-96	97-248		249-252	253	254	255-256

Figure 13. Format-E DSCB

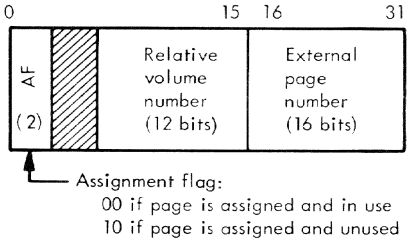


Figure 14. External Page Entry

List of volumes for private data set (each 6 bytes, variable length) ctd from Format-E DSCB if required / External page entries each 4 bytes long (on full-word boundaries)	External page entries, each 4 bytes long (on full-word boundaries)	Pointer to next DSCB if chained	DSCB type	DSCB ID	Check sum
1-248		249-252	253	254	255-256

Figure 15. Format-F DSCB

Physical Sequential Data Sets

Each private storage volume that is formatted for the physical sequential access method has a volume table of contents (VTOC) that describes its contents; the VTOC contains all data set control blocks (DSCBs) for the data sets contained on that volume. The VTOC may be located anywhere on the volume, starting on a track boundary. It may vary in size, but always has an integral number of tracks. The starting address of the VTOC is recorded in the standard volume label (refer to Figure 16).

The standard volume label resides on cylinder 0, track 0, of the volume. When the volume enters the system, the standard volume label and the VTOC are created. All space on the volume (except the space occupied by the volume label and VTOC) is then available for allocation.

The format of VTOC is specified when the volume enters the system. All records have a 44-byte key and a 96-byte data portion. Each of these records becomes a DSCB, of varying type, and describes the attributes and extents of a data set.

Each DSCB contains the name, description, and location on the volume of its associated data set. It is created by the system when the data set is stored on the volume. The DSCB serves as the data set's label and contains information similar to magnetic-tape labels.

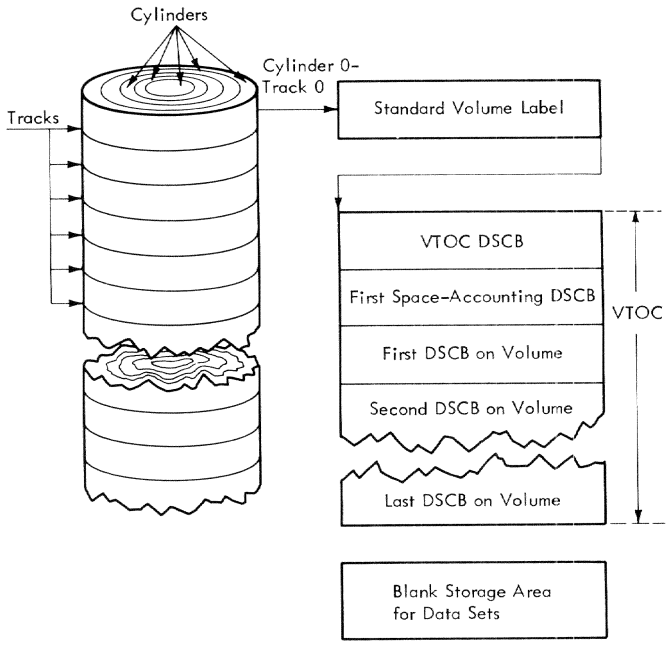


Figure 16. Direct Access Labels for Physical Sequential Data Sets

The DSCBs are entered into the VTOC as they are created and are placed in the first available space, starting with the VTOC-DSCB (a format-4 DSCB). Available DSCB plots are recognized by a key field of binary 0s. When a data set is deleted, its DSCB is overwritten with 0s, making it a format-0 DSCB. As available extents increase, more direct access device storage management (DADSM) DSCBs are entered into the first available slot. At any time, the VTOC has a mixture of formats-1, -3, -4, and -5 DSCBs, and "holes" for format-0 DSCBs. These formats will be explained below.

DSCBs in formats -3 and -5 will appear to have a key length of 44 bytes, but portions of the key may actually contain data. The DSCBs are all assigned the same format to provide the flexibility to convert an available DSCB (format-0) to another type of DSCB, and back again, without the necessity for changing the format or modifying the channel programs that act upon them.

Standard Volume Label (Figure 17): Always the third record on cylinder 0, track 0, of the volume; this label identifies the volume. The fields in the label are the same as those in the magnetic tape volume label, described in Table 4, except for bytes 12-21, which contain the address of the VTOC.

Note: System programmers using the Time Sharing Support System (TSSS) must expect an 84 byte volume label; the label as described in Figure 17, will be prefixed by a key of characters 'VOL1' and the fields will be displaced by 4 bytes.

Format-0 DSCB: This is the standard format of a data record in the VTOC that is not currently occupied by any other format of DSCB. The key and data portions contain binary 0s.

Format-1 DSCB (Figure 18): This format is common to all physical sequential data sets. It consists of a 44-byte key field and a 96-byte data field. The format-1 DSCB is the data set label for direct access volumes; it corresponds to a tape header label. Also, it describes up to three sets of contiguous tracks or cylinders on which the data set resides.

Format-3 DSCB (Figure 19): This format is used to describe extra extents of a data set, if there are more than can be described in the format-1 DSCB. If additional space is needed, this DSCB will point to another format-3 DSCB.

Format-4 DSCB (Figure 20): This format is the first DSCB in the VTOC of physical sequential volumes.

VOL (label identifier)	1 (volume label number)	Volume serial number	Volume security	VTOC pointer	Reserved (currently blank)	Owner name-and-address code	Reserved (currently blank)
1-3	4	5-10	11	12-21	32-41	42-51	52-80

Figure 17. Standard Volume Label (Physical Sequential Data Sets on Direct Access)

Data set name	F1 (hexa-decimal) format identifier	Data set serial number	Volume sequence number	Creation date	Expiration date	Number of extents on volume	Not used	Spare	System code	Reserved
1-44	45	46-51	52-53	54-56	57-59	60	61	62	63-75	76-82

Data set type	Record format	Option code	Block length	Record length	Key length	Key location	Data set indicator	Original request for space	Secondary allocation	Last record pointer
83-84	85	86	87-88	89-90	91	92-93	94	95	96-98	99-103

Spare	Extent type indicator	Extent sequence number	Lower limit	Upper limit	Additional extents (same as bytes 106-115)	Additional extents (same as bytes 106-115)	Pointer to next DSCB record
104-105	106	107	108-111	112-115	116-125	126-135	136-140

Figure 18. Format-1 DSCB

03030303 (hexadecimal)	Extent (in key) — 4 groups of fields similar to bytes 106-115 in format-1 DSCB	F3 (hexadecimal) format identifier	Additional extents; 9 groups of fields similar to bytes 106-115 in format-1 DSCB	Pointer to next format-3 DSCB
1-4	5-44	45	46-135	136-140

Figure 19. Format-3 DSCB

Key field (contains hexadecimal 04s)	F4 (hexadecimal) format identifier	Last active format-1 or format-A DSCB	Available DSCB records	Next available alternate track	Number of available alternate track	VTOC indicators	01 (hexadecimal) number of extents
1-44	45	46-50	51-52	53-56	57-58	59	60

	Reserved	Device constants	Spare	Gross available space	Pointer to format-6 DSCB	VTOC extent - same as bytes 106-115 in format-1 DSCB	Spare
	61-62	63-76	77-95	96-100	101-105	106-115	116-140

Figure 20. Format-4 DSCB

05050505 (hexadecimal) Key identification	Available extent			Available extents (in key) - same form as bytes 5-9	F5 (hexadecimal) format identifier	Available extents (same format as bytes 5-9)	Pointer to next format-5 DSCB
	Relative track address	Number of full cylinders	Number of additional tracks				
1-4	5-6	7-8	9	10-44	45	46-135	136-140

Figure 21. Format-5 DSCB

Format-5 DSCB (Figure 21): This format is always the second VTOC-DSCB for a volume containing physical sequential data sets. It describes available extents on the volume. If additional extents are needed, this DSCB is chained to other format-5 DSCBs.

MAGNETIC TAPE VOLUMES

Magnetic tapes may be unlabeled or have standard labels. The control program supplies routines for automatic positioning and volume switching of such volumes.

All standard tape labels are 80-character records, written in extended binary coded decimal interchange code (EBCDIC) on nine-track tape units, and in binary coded decimal (BCD) or the American National Standard Code for Information Interchange, ANSI X3.4-1968 (hereafter referred to as ASCII) on seven-track units. The tape label is recorded in the same density as the data on the tape, specified in the DDEF command.

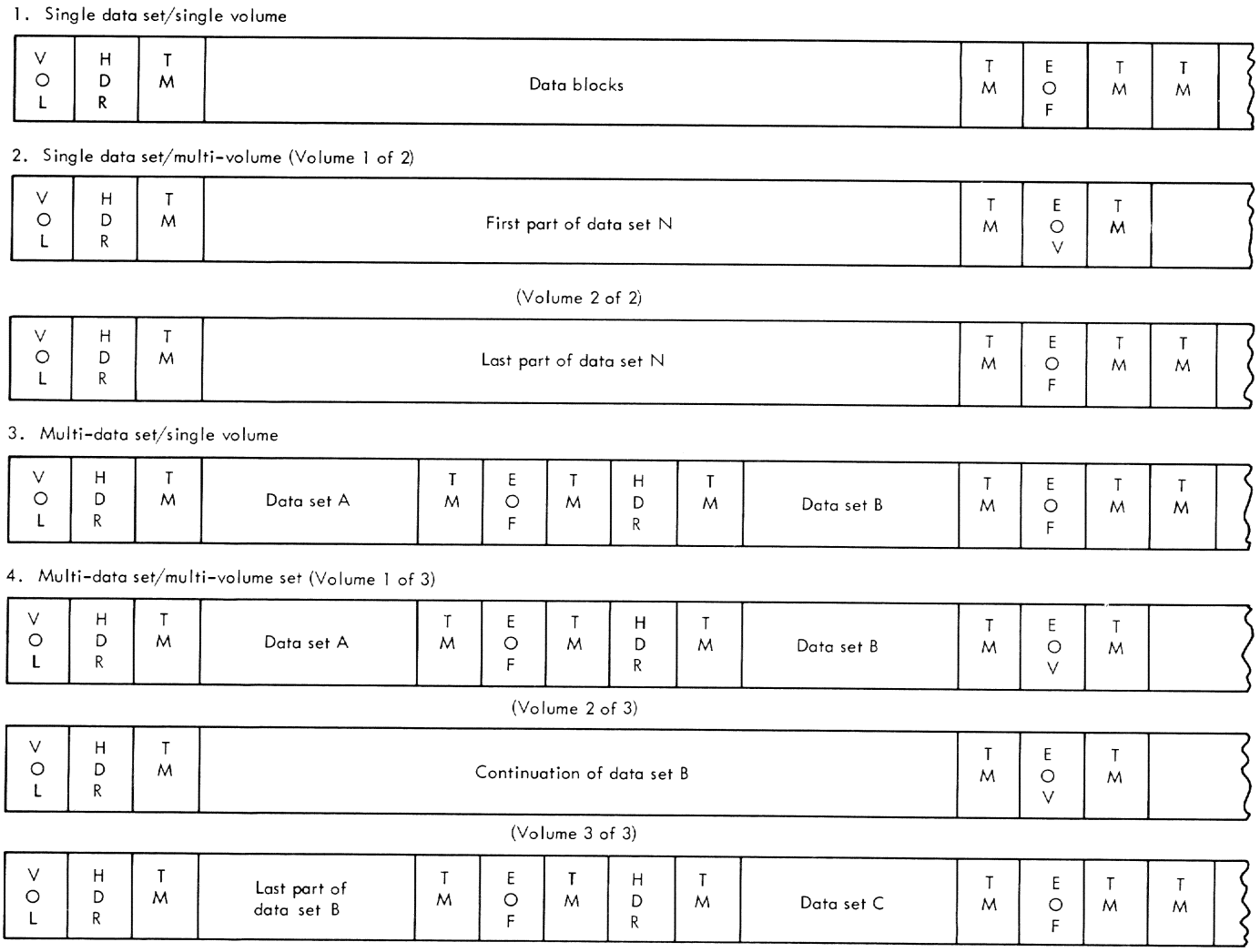
Standard Tape Organization

The organizations of standard labels and data on magnetic tape, for the tape organizations below, are illustrated in Figure 22.

Single Data Set/Single Volume: The volume begins with a volume label. The data set begins with a data set header label, a user header label (optional), and a tape mark. The entire content of the data set is next. The last data block is followed by a tape mark and an EOF trailer label group, which is followed by the two tape marks that are the last records on the volume.

Single Data Set/Multi Volume: This is a simple expansion of the preceding description, where the amount of information requires more than one volume. All volumes, except the last, of the set will contain the same organization as for a single data set/single volume, except that the trailer will be of an EOV trailer label group. The last volume will have the same organization, except that the trailer group will be an EOF trailer label group, followed by two tape marks.

Multi-Data Set/Single Volume: The volume begins with a volume label. Every data set will start with a data set header label, a user header label (optional), and one tape mark. The data set follows. Every data set (except the last) will conclude with a tape mark, an EOF trailer label group, and another tape mark. The last data set is the same, except that the EOF trailer label group is followed by two tape marks.



Note: This is an example (Volumes 1 through 3, inclusive) of the successive recording of data sets on physical volumes to maximize tape use.

Figure 22. Standard Label and Data Organization on Magnetic Tape

Multi-Data Set/Multi Volume: This volume is similar to the previous one, except that the amount of information requires more than one volume. These rules must be followed in producing the additional volumes:

1. Each volume, except the last, must conclude with a tape mark, an EOV trailer label group, and a tape mark.
2. Each volume begins with a volume label.
3. The initial data set header label on each volume, except the first, is a repetition of the last data set header label on the previous volume, with the exception of the volume sequence number.

Volume Label

The volume label identifies a volume and its owner, and is used to verify that the correct volume is mounted. It can also be used to prevent use of the volume by authorized programs.

Tables 5 and 6 show the organization of standard tape labels and describe their fields.

A tape using standard labels is identified as such by the system when it reads the initial record, and determines that it is a volume label by finding that these criteria have been met:

- Initial record consists of 84 characters

- First four characters of the record are VOL1

The system automatically checks the volume label to ensure that it is in the proper format; if the format is correct, the system checks the label information. Should the check indicate an error (for example, the system finds that the wrong volume has been mounted), it issues a message to the operator. Similarly, messages are issued if errors are detected in other label and format checks.

Data Set Header Label Group

The data set header label group consists of HDR1 and HDR2. These labels are created by the system when the data is recorded. HDR1, as shown in Tables 7 and 8, contain system data and device-dependent information. HDR2, shown in Tables 9 and 10, contain data set attributes. If there are no user header labels, HDR2 is followed by a tape mark. The group can be used in forward-reading operations to:

- Locate the data set
- Verify reference to the data set
- Provide information for the DCB.

User Header Label Group

A maximum of eight user header labels may follow the data set header label group. The labels are written by the system, as directed by the problem program that records the tape. The group is ended by a tape mark.

When the tape is read, the user header label group is made available to the problem program by the system; the format of a label is shown in Table 11 .

Data Set Trailer Label Group

The data set trailer label group consists of two labels that duplicate the data set header labels to facilitate backward reading of the tape. The format for the trailer labels is identical to the data set header labels, except for the fields shown in Table 12. These labels duplicate the data set header labels to facilitate backward reading of tape. Location and verification 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, optionally, follow the data set trailer label group. They are written exactly

Table 5. EBCDIC Volume Label Format (Magnetic Tape)

Field Number	Position (Bytes)	Name	Use
1	1-3	Volume label identifier	Contains "VOL" to indicate record is volume label
2	4	Volume label number	Contains "1" to indicate this is first volume label
3	5-10	Volume serial number	Contains unique identification code, assigned when volume entered system; it can be copied on external surface of reel for visual identification; field normally numeric (000001-999999), but may contain any six alphameric characters
4	11	Not used by TSS	Recorded as EBCDIC 0s
5	12-21	Reserved	Recorded as blanks
6	22-31	Reserved for manufacturers	Reserved for future use; must be blank
7	32-41	Reserved	Reserved for future use; must be blank
8	42-51	Owner name-and-address code	Contains unique identification of owner of volume
9	52-80	Reserved	Reserved for future use; must be blank

the same as the user header labels, except for the difference noted in Table 13.

User trailer labels specify information pertaining to the data sets on the volume. These labels contain information that can not be put into the standard header labels or into the records themselves.

A common use of these labels is to store control information (for example, the number of records in the data set, or the number of read-errors encountered in reading the data set).

Table 6. ASCII Volume Label Format (Magnetic Tape)

Field Number	Position (Bytes)	Name	Use
1	1-3	Volume Label Identifier	Contains "VOL" to indicate record is volume label
2	4	Volume Label Number	Contains "1" to indicate this is first volume label
3	5-10	Volume Serial Number	Contains six alphameric characters permanently assigned by the owner to identify this volume
4	11	Volume Accessibility	An alphameric character indicating restrictions on access to the volume. A blank indicates unlimited access; any other character indicates special handling according to the agreement between interchange parties.
5	12-31	Reserved for future standardization	Must contain blanks
6	32-37	Reserved for future standardization	Must contain blanks
7	38-51	Owner Identification	Contains unique ID of owner of volume
8	52-79	Reserved for future standardization	Must contain blanks
9	80	Label Standard Level	May contain a "1" or blank. One (1) indicates volume labels and data formats conform to this standard. Blank indicates labels and data formats require agreement of interchange parties.

Table 7. EBCDIC Data Set Header-1 Label Format

Field Number	Position (Bytes)	Name	Use
1	1-3	Label identifier	Contains "HDR" to indicate this is header label
2	4	Data set label number	Contains "1" to indicate this is first data set header label
3	5-21	Data set identifier	Identifies data set; may contain only alphameric characters
4	22-27	Data set serial number	Contains same identification code as in field #3 of the initial volume label of the volume on which the data set resides or of the first volume of a multidata set aggregate
5	28-31	Volume sequence number	Indicates volume on which data set is recorded, relative to volume on which data set or aggregate begins
6	32-35	Data set sequence number	Indicates position of data set relative to first data set in aggregate; range from 0001 to 9999
7	36-39	Generation number	Indicates generation number (0001-9999) of data set
8	40-41	Version number of generation	Indicates version of generation of data set
9	42-47	Creation date	Indicates year and day data set was created; recorded as bYYDDD (YY is 00-99 and DDD is 001-366)
10	48-53	Expiration date	Indicates first day tape may be over-written; recorded as bYYDDD
11	54	Not used by TSS	Contains EBCDIC 0s
12	55-60	Unused	Contains 0s
13	61-73	System code	Contains unique identification of programming system
14	74-80	Reserved	Reserved for future use; must be blank

Table 8. ASCII Tape Data Set Header-1 Label Format

Field Number	Position (Bytes)	Name	Use
1	1-3	Label identifier	Contains "HDR" to indicate this is header label
2	4	Data set label number	Contains "1" to indicate this is first data set header label
3	5-21	Data set identifier	Identifies data set; may contain only alphameric characters
4	22-27	Data set serial number	Contains same identification code as in field #3 of the initial volume label of the volume on which the data set resides or of the first volume of a multidata set aggregate
5	28-31	Volume sequence number	Indicates volume on which data set is recorded, relative to volume on which data set or aggregate begins
6	32-35	Data set sequence number	Indicates position of data set relative to first data set in aggregate; range from 0001 to 9999
7	36-39	Generation number (optional)	Indicates generation number (0001-9999) of data set
8	40-41	Version number of generation (optional)	Indicates version of generation of data set
9	42-47	Creation date	Indicates year and day data set was created; recorded as bYYDDD (YY is 00-99 and DDD is 001-366)
10	48-53	Expiration date	Indicates first day tape may be over-written; recorded as bYYDDD
11	54	Data set accessibility	Contains an alphameric indicating access restrictions to this data set. A blank indicates unlimited access; any other character indicates special handling according to agreement between interchange parties.
12	55-60	Block count	Must contain zeros
13	61-73	System code (optional)	Contains unique identification of programming system
14	74-80	Reserved	Reserved for future use; must be blank

Table 9. EBCDIC Data Set Header-2 Label Format

Field Number	Position (Bytes)	Name	Use
1	1-3	Label identifier	Contains "HDR" to indicate header label
2	4	Data set label number	Contains "2" to indicate second data set header label
3	5	Record format	Indicates format of records: F - Fixed length V - Variable length U - Undefined
4	6-10	Block length	Indicates length of block; interpretation of field depends on format specified in field #3: Format F - Length of a physical block Format V - Maximum length of a block Format U - Maximum length of a block
5	11-15	Logical record length	Indicates length of logical record; interpretation depends on record format specified in field #3: Format F - Length of a logical record Format V - Maximum logical record length Format U - This field contains 0s
6	16	Density	Indicates tape density: Density (bits/inch) Model 2400 DEN Value: 0 — 7-track: 200 — 9-track: --- 1 — 556 — --- 2 — 800 — 800 3 — --- — 1600 4 — --- — 6250
7	17	Data set position	Indicates whether volume switched previously for data set; volume switched, 1 is written; if not, 0 is written; when tape is read backwards, this information indicates when volume switch is required
8	18-34	Not used by TSS	Must be blank
9	35-36	Tape recording technique	Indicates, for 7-track tape, technique in creating this data set BCD Code — Meaning Cb — Data conversion feature used Eb — Even parity used Tb — BCD to EBCDIC translation required ET — Even parity used; BCD to EBCDIC translation required bb — Odd parity, no translation required; or this is 9-track tape

Table 9. EBCDIC Data Set Header-2 Label Format (continued)

10	37	Control character	Indicates type of control character (first data character of each record) used to control printer-spacing and punch-selection: A - American National Standard FORTRAN control character M - Machine-code control character b - No control character used
11	38-80	Reserved	Must be blank

Table 10. ASCII Data Set Header-2 Label Format

Field Number	Position (Bytes)	Name	Use
1	1-3	Label identifier	Contains "HDR" to indicate header label
2	4	Data set label number	Contains "2" to indicate second data set header label
3	5	Record Format	Indicates format of records: F - Fixed length D - Variable length (specified in decimal) U - Undefined
4	6-10	Block length	Contains five numeric characters specifying the maximum number of characters per block
5	11-15	Logical record length	Indicates length of logical record; interpretation depends on record format specified in field #3: Format F - Length of logical record Format D - Maximum logical record length (count fields incl.) Format U - Contain zeros
6	16-50	Reserved for operating systems	Contains alphameric characters reserved for operating systems use
7	51-52	Buffer offset (optional)	Contains two numeric characters specifying the length, in characters, of any additional field inserted before a data block, e.g., block length
8	53-80	Reserved for future standardization	Must contain blanks

Table 11. User Header Label Format

Field Number	Position (Bytes)	Name	Use
1	1-3	Label identifier	Characters "UHL" indicate this is user header label
2	4	Label number	Identifies relative position (1-8) of label within label group
3	5-80	User specified	Used to specify information pertaining to data set or sets on volume

Table 12. Data Set Trailer Label Format

Format of trailer labels is identical to data set header labels (Tables 5 and 6), except for these fields:

Field Number	Position (Bytes)	Name	Use
1	1-3	Label identifier	Characters "EOV" indicate end of volume; "EOF" indicates end of data set; field indicates this is data set trailer label
2	4	Label number	Indicates label is first (1) or second (2) data set trailer label
12	55-60	Block count	Indicates number of blocks in data set on current volume of multi-volume data set, with range of 000000 to 999999; indicates number of blocks from last label of label header group to first label of trailer label group, exclusive of tape marks

Table 13. User Trailer Label Format

Field Number	Position (Bytes)	Name	Use
1	1-3	Label identifier	Characters "UTL" indicate this is user trailer label
2	4	Label number	Identifies relative position (1-8) of label within label group
3	5-80	User specified	Used to specify information pertaining to data sets on the volume

DATA SET DEFINITION RULES FOR LANGUAGE PROCESSING

Table 14 provides information relating to the organization of and DDEF requirements for data sets involved in assembly, compilation and linkage-editing.

DATA SET DEFINITION RULES FOR TSS COMMANDS

Table 15 provides information relating to the structure of and DDEF requirements for data sets processed by TSS commands.

Table 14. Data Set Definition Rules for Language Processing

Command	Related Data Sets	DSORG	Data Set Definition Rules
ASM (ASSEMBLER)	Source program data set.	VI Line data set	Source program data sets: If supplied as part of SYSIN data set, these data sets do not require any further definition. If supplied as prestored data sets, they must be cataloged. No DDEF required prior to an ASM, COBOL, FTN, FTNH, HASM, PLI or PLIOPT command.
	Object module.	VS (VP member)	
	Listing data set.	VI List data set	
COBOL (COBOL)	Source program data set.	VI VS	
	Object module.	VS (VP member)	
	Listing data set.	VI List data set	
	Load data set (created by the program product COBOL not yet converted to object module).	VS	
	Source statements for insertion by preprocessor (Note 1).	VI (VP member)	
FTN (FORTRAN)	Source program data set.	VI Line data set	Object module: The module is placed in the library at the top of the program library list. If a job library is to receive the object module, a DDEF command must define the library.
	Object module.	VS (VP member)	
	Listing data set.	VI List data set	
FTNH (FORTRAN H EXTENDED)	Source program data set.	VI VS	
	Object module.	VS (VP member)	
	Listing data set.	VI List data set	
	Load data set (created by the program product FORTRAN H not yet converted to object module).	VS	Listing: No DDEF command required

Note 1: DDEF command is required if VP data set is not USERLIB.

Table 14. Data Set Definition Rules for Language Processing (Continued)

Command	Related Data Sets	DSORG	Data Set Definition Rules
HASM (ASSMBLR H)	Source program data set.	VI VS	Same rules as for ASM.
	Object module.	VS (VP member)	
	Listing data set.	VI List data set	
	Load data set (created by the program product ASSEMBLER H not yet converted to object module).	VS	
PLI (PL/I (F))	Source program data set.	VI Line data set	Same rules as for ASM.
	Object module.	VS (VP member)	
	Listing data set.	VS List data set	
	Load data set (an object module that has been created by the TSS PL/I (F) compiler but not converted from card-image form).	VS	No DDEF command required.
	Source statements for insertion by preprocessor.	VI (VP member)	DDEF command required if VP data set is not USERLIB.
	Storage for: 1. Translated source statements when 48-character set not used. 2. Source statements generated by the preprocessor.	VI Line data set	DDEF command required if data set is not the MAC.name(0) data set created automatically by the preprocessor.
PLIOPT (PL/I OPTIMIZING COMPILER)	Source program data set.	VI VS	Same rules as for ASM.
	Object module.	VS (VP member)	
	Listing data set.	VI List data set	
	Load data set (created by the program product PL/I OPTIMIZING COMPILER not yet converted to object module).	VS	
	Source statements for insertion by preprocessor.	VI (VP member)	DDEF command required if VP data set is not USERLIB.

Table 14. Data Set Definition Rules for Language Processing (Continued)

Command	Related Data Sets	DSORG	Data Set Definition Rules
LNK (LINKAGE EDITOR)	Source program data set.	VI Line data set	Same rules as for ASM, FTN, and PL/I (F).
	Libraries that are to supply object modules.	VP	Each library referred to by INCLUDE statements except USERLIB and each job library used by automatic call must be defined by a DDEF command.
	Library to receive output object module.	VP	If library at top of program library list is to receive output object module, no additional DDEF in this task. If another library is to receive output, it must be defined by previous DDEF command and be specified by its ddname to linkage editor program.
	Listing data set.	VI List data set	No DDEF command required.

Table 15. Data Set Definition Requirements for Commands

Command	Related Data Sets	DSORG	Data Set Definition
BACK	New SYSIN data set that is to control completion of this task in nonconversational mode.	VS, VI	New SYSIN data set must be cataloged or defined by previous DDEF command in conversational portion of this task.
BUILTIN	Object module in user's program library hierarchy.	VP	Data set must be defined in current task, or must be cataloged.
CATALOG	Data set to be cataloged.	PS	Data set to be cataloged must be defined by previous DDEF command in this task, unless UPDATE option specified.
CDD	Data set containing only DDEF commands.	VI	Data set must be cataloged, defined in current task.
CDS	Original data set: Existing data set or one or more members of partitioned data set.	VS, VI	Data set to be copied must be cataloged or defined by previous DDEF command in this task.
	New copy: Can be data set, one member of partitioned data set, or entire partitioned data set.	VS, VI	Provided by system.
CLOSE	Data set to be closed.	any	Data set to be closed must be defined by previous DDEF command in this task.

Table 15. Data Set Definition Requirements for Commands (continued)

Command	Related Data Sets	DSORG	Data Set Definition
DATA[1]	Data set to be entered.	VS, VI	No DDEF command is required if the data set is to reside on public storage; data follows this command in input stream. If the data set is to reside on private storage a DDEF must be issued before the command.
DEFAULT	User profile data set SYSPRX in USERLIB.	VP	Provided by system.
DELETE	Data set whose name is to be removed from catalog.	Any	No DDEF command required for this command.
DSS?	Data sets whose status is desired.	Any	Each data set whose status is to be presented must be cataloged; no DDEF command required for this command.
DUMP	Data set to be printed as a result of program control command DUMP.	VI List data set	DDEF command whose ddname is PCSOUT must be defined prior to execution of DUMP command.
EDIT[2]	Data set to be processed by the Text Editor.	VI	Data set must be cataloged, or defined in current task. This is done automatically.
END[2]	Data set being processed by the Text Editor, or indicates PROCDEF command completion.	VI	No DDEF command required for this command.
ERASE	Data set to be erased.	Any	Data set to be erased must be cataloged, or DDEFed.
EVV	Private data sets whose names are to be entered in catalog.	VS, VI, VP	No DDEF command required for this command.
EXECUTE	SYSIN data set for nonconversational task set up by this command.	VS, VI	Data set must be cataloged; no DDEF command required by this command.
LINE?	Line data set containing lines to be presented.	VI List or line data set	Line data set must be cataloged or defined by previous DDEF command in this task.
LOAD	Object module to be loaded.	VP (VS member)	Object module to be loaded is identified by external name specified in this command; it must be in a library in the current program library list.
MODIFY	Data set to be changed.	VI	Data set must be cataloged or defined by previous DDEF command in this task.
PC?	Data set whose status is required.	Any	Each data set whose status is to be presented must be cataloged; no DDEF command required for this command.
PERMIT	Data sets for which sharing is permitted	Any	Data sets for which sharing is permitted must be cataloged; no DDEF command required for this command.

Table 15. Data Set Definition Requirements for Commands (continued)

Command	Related Data Sets	DSORG	Data Set Definition
POD?	Virtual partitioned data set for which information about its members is given.	VP	Virtual partitioned data set must be cataloged, or defined by previous DDEF command in this task.
PRINT	Data set to be printed.	PS, VS, VI	Data set must be cataloged or defined by previous DDEF command in this task. A previous DDEF required for unlabeled tapes.
PROCDEF	Data set which consists of other commands, to become a user-written procedure.	VI	Provided by system.
PROFILE	User profile data set in USERLIB, session profile in task virtual memory.	VP	Provided by system.
PUNCH	Data set to be punched on cards.	VS, VI	Data set must be cataloged or be defined by previous DDEF command in this task.
REGION[2]	Data set to be processed by the Text Editor.	VI	Data set must be cataloged, or defined in current task.
RELEASE	Data set whose definition is to be released.	Any	Data set whose definition is to be released must be defined in previous DDEF command in this task.
RET	VAM data set whose data set descriptor is to be changed.	VS, VI, VP	Data set must be cataloged.
SHARE	Data sets for which sharing is requested.	Any	Data sets for which sharing is requested must be cataloged by their owner; no DDEF command required by this command.
SYNONYM	User profile data set in USERLIB, session profile in task virtual storage.	VP	Data sets must be defined in current task.
TV	Physical sequential data set (from a VT operation) to be written on a VAM volume.	PS	Data set (input) must be cataloged or defined in current task.
VT	VAM data set to be copied to magnetic tape as a physical sequential data set.	VS, VI, VP	Data set (input) must be cataloged or defined in current task.
VV	VAM data set to be copied into direct access storage.	VS, VI, VP	Data set (input) must be cataloged or defined in current task.
WT	Data set to be recorded on magnetic tape in print format.	VS, VI	Data set must be cataloged or defined by previous DDEF command in this task.

[1]If the DATA command was used to create the data set within the current task, then the data set is defined as if a DDEF command had been issued by the user directly. If the data set is also VAM organized and resides in public storage, it is automatically cataloged.

[2]These are the basic directive commands of the Text Editor. See Command System User's Guide for details concerning the data manipulation commands of this facility.

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

The prime consideration in the selection of a record format is the nature of the data set itself. The user knows the type of input his program will receive and the type of output it will produce. 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 and write the data set.

In the case of ASCII tape records, the user should be aware that TSS translates the records to EBCDIC on input to process them and translates them back to ASCII form for output. Since some ASCII records begin with a control information field that is foreign to TSS, the size of this field (buffer offset) must be identified as part of the record format.

The record format of a data set is placed into the data control block according to specifications in the DCB macro instruction, the DDEF command, or the DDEF macro instruction.

FIXED-LENGTH (FORMAT-F)

Format-F records are fixed-length. If unblocked format F, the logical record constitutes the block. If blocked format-F (applicable to BSAM and QSAM only), the number of logical records within a block (blocking factor) is normally constant for every block in the data set, unless the block is truncated (short block).

The system performs physical length checking on format-F records, automatically making allowances for truncated blocks.

VARIABLE-LENGTH (FORMAT-V AND FORMAT-D)

Format-V and format-D (ASCII tape only) records are variable-length records, each of which describes its own length. When blocked (applicable to BSAM and QSAM only), each block also includes its block length. The system performs length checking of the records and blocks.

The first four characters of the record contain control information describing the length of the record; the format of this information depends on whether the record is part of a virtual storage data set or a physical sequential data set.

When unblocked, the logical record and the block control information constitute the block. The block control information (four bytes) must be included in the record length.

In blocked format-V, the block length, LLbb, is prefixed to each block, LL represents the block length, and bb represents two characters reserved for system use. This four-byte block length field must be included in the block length.

Variable length records on ASCII tapes are specified as format-D. They contain the same control information as format-V records, but this information is recorded in decimal characters.

UNDEFINED-FORMAT (FORMAT-U)

Format-U is provided to permit the processing of any blocks that do not conform to the F or V formats. Since each block is treated as a logical record (unblocked), any deblocking must be done by the user's program.

CONTROL CHARACTER

The user may optionally specify, in the DDEF command, the DDEF macro instruction or the DCB macro instruction, that a control character precedes each logical record in a data set, as shown in Figure 23. This control character specifies carriage control when the data set is printed, or stacker selection when the data set is card-punched. The character itself is never printed or punched, but it is a part of the record in storage.

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

DIAGRAMS OF RECORD FORMATS

The following pages show the standard external record formats for TSS. An external format -- the format seen by the user -- may differ from the internal format.

- Record formats for virtual sequential data sets are shown in Figure 24.

- **Record formats for virtual index sequential data sets are shown in Figure 25.**
- **Record formats for physical sequential data sets are shown in Figures 26 and 27.**
- **Record formats for physical sequential data sets on ASCII tapes are described in Figures 28 and 29.**
- **Virtual partitioned data sets must conform to the record formats shown in Figure 24 for virtual sequential members, and to those shown in Figure 25 for virtual index sequential members.**

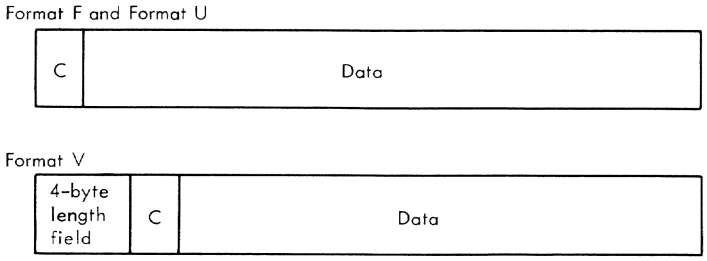


Figure 23. Placement of Control Character in a Record

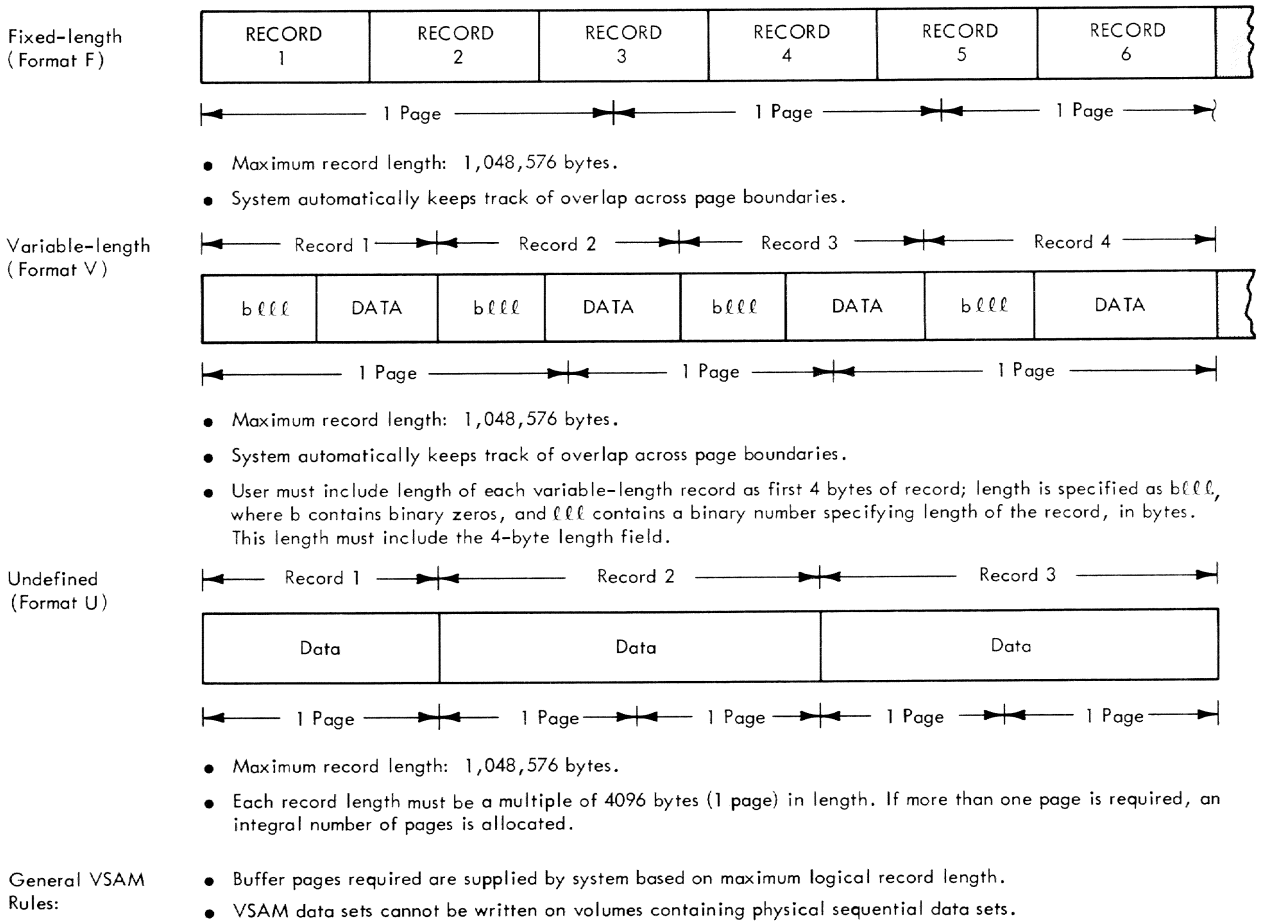
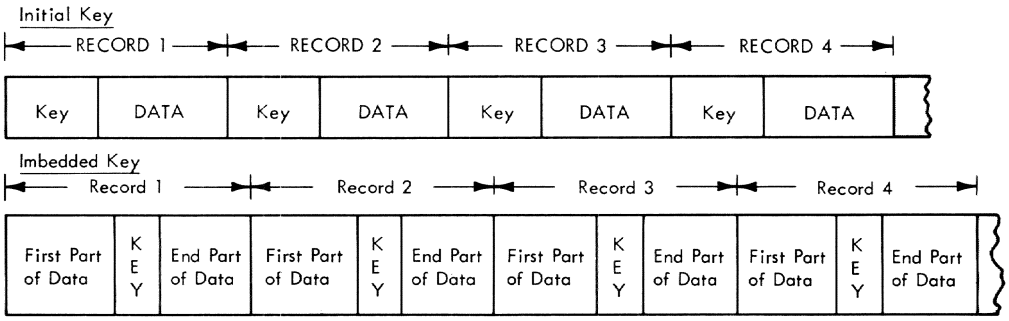
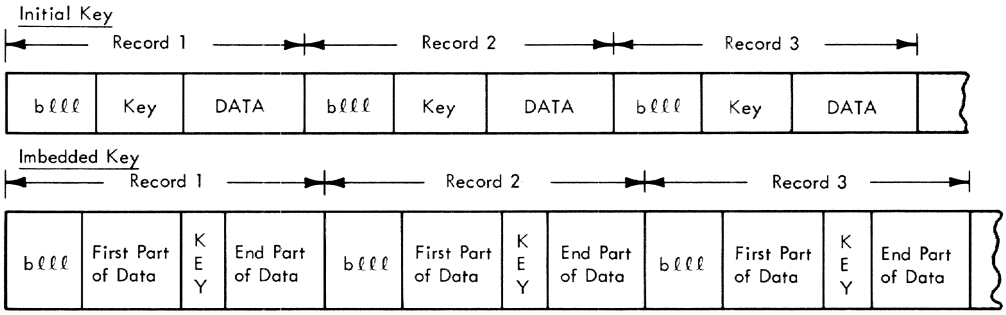


Figure 24. Record Formats -- VSAM

Fixed-length
(Format F)

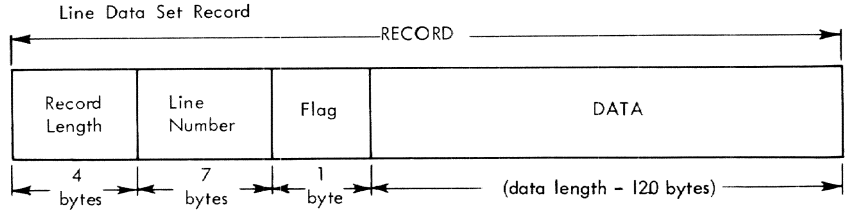


Variable-length
(Format V)



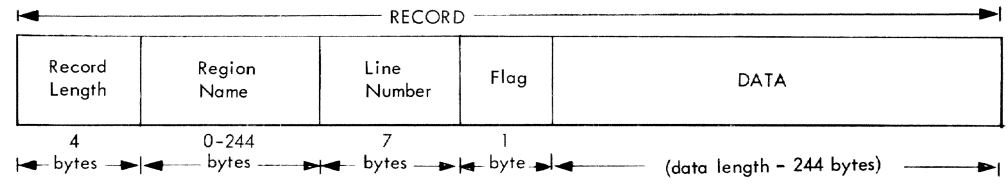
- Maximum logical record length: 4000 bytes.
- Maximum number of records per data page: 1300.
- Maximum key length: 255 bytes.
- Maximum number of data pages: 65,000.
- Maximum number of overflow pages: 240.
- Maximum number of records per overflow page: 255.
- Maximum number of directory pages: 255.
- User must include length of each variable-length record as first 4 bytes of record; length is specified as bℓℓℓ, where b contains binary zeros, and ℓℓℓ contains a binary number specifying length of the record, in bytes. This length must include the 4-byte length field.

Line Data Set Record



- Maximum record length: 132 bytes.
- Maximum data length: 120 bytes.
- Flag byte indicates whether record originally came from terminal keyboard (01) or card reader (00).

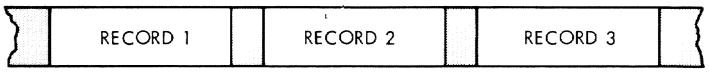
Region Data Set Record



- Maximum record length: 256 bytes.
- Maximum data length: 244 bytes.
- Flag byte indicates whether record originally came from terminal keyboard (01) or card reader (00).

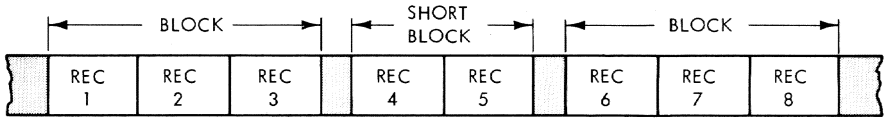
Figure 25. Record Formats -- VISAM

Fixed-length
(Format F)



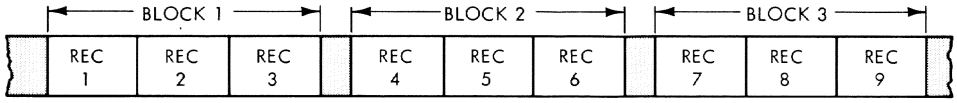
- Maximum record length – 32,760 bytes.
- Each block treated as a logical record.

Fixed-length
Blocked
(Format FB)



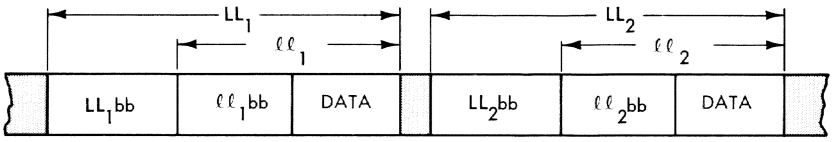
- Maximum block length – 32,760 bytes.
- Blocking factor is usually constant; however, data set may contain truncated or short blocks.

Fixed-length,
Blocked
Standard Blocking
(Format FBS)



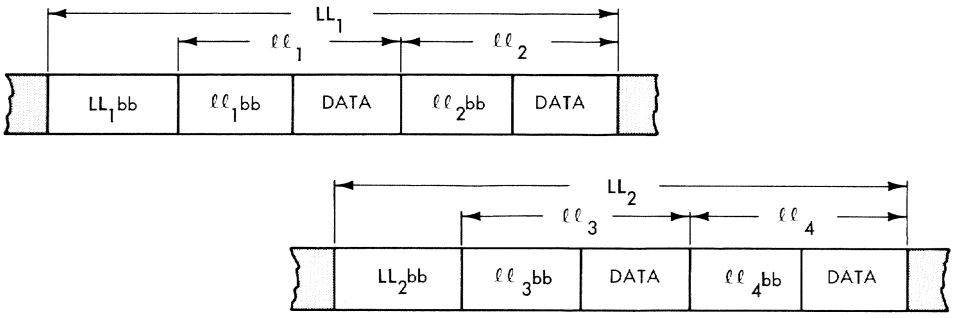
- Maximum block length – 32,760 bytes.
- Last block may be truncated; truncated block invokes end-of-volume routines.

Variable-length
(Format V)



- Maximum logical record length – 32,756 bytes.

Variable-length,
Blocked
(Format VB)



- Maximum logical record length – 32,763 bytes.
- Each logical record must describe its own length; this information must be included by user as first 4 bytes of each record:

  ℓℓ - Binary number specifying record length in bytes.

  bb - Binary 0s.

- System performs length checking of blocks containing Format-V records, based on user-supplied length information; when data sets with Format-V records (either blocked or unblocked) are created, a 4-byte control block is required in the form LLbb, where:

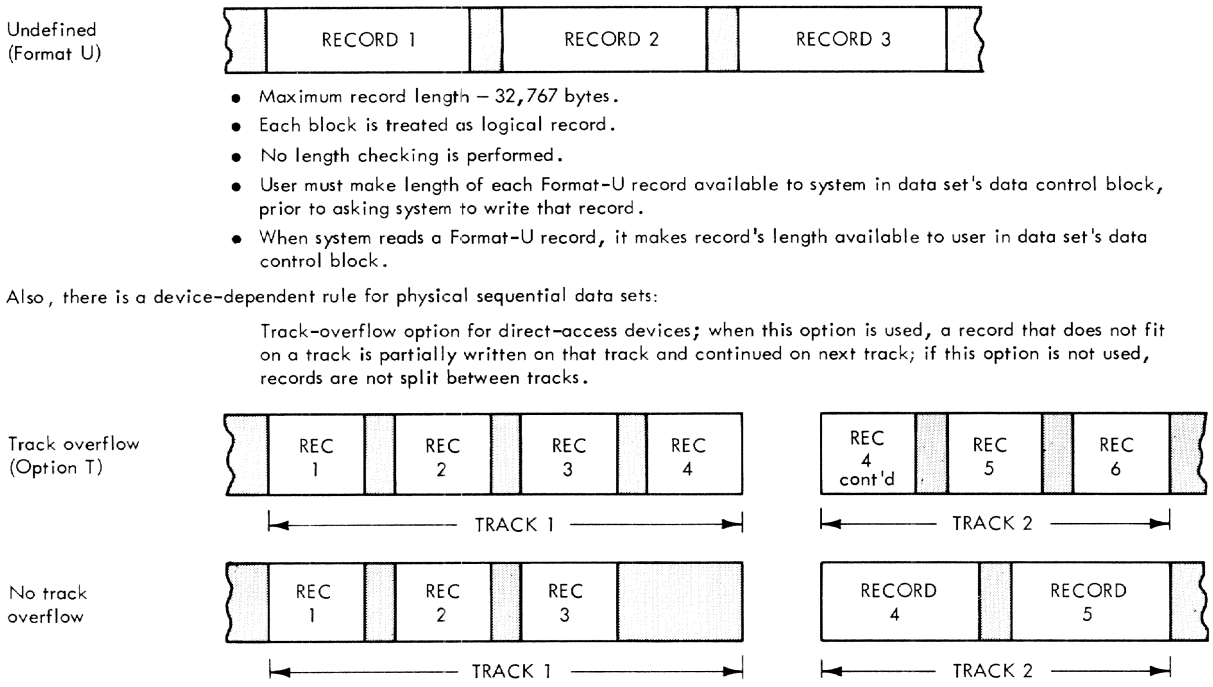
  LL - Binary number specifying block length in bytes.

  bb - Two bytes reserved for system use.

  Value of LL is determined by adding the ℓℓ s of the records within block and adding 4 bytes for the control field.

- Format-V and Blocked Format-V records cannot be processed on 7-track tape units without data conversion feature.

Figure 26. Record Formats -- Physical Sequential Data Sets Without Keys

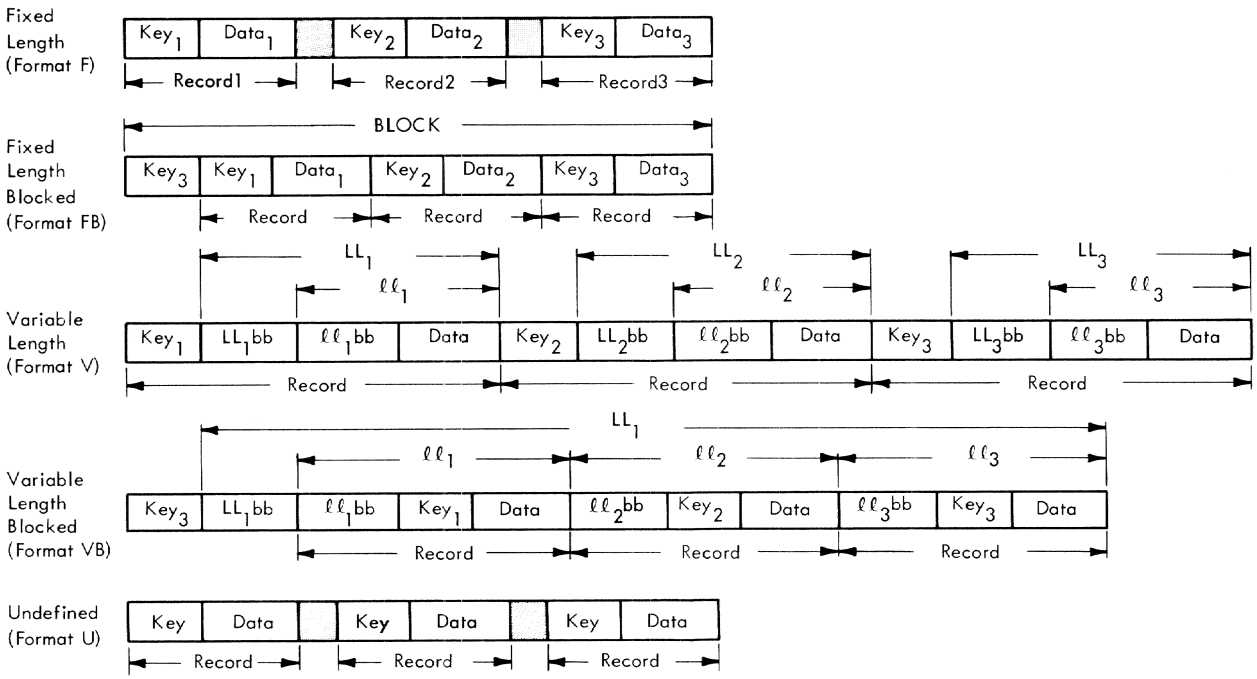


- Maximum record length – 32,767 bytes.
- Each block is treated as logical record.
- No length checking is performed.
- User must make length of each Format-U record available to system in data set's data control block, prior to asking system to write that record.
- When system reads a Format-U record, it makes record's length available to user in data set's data control block.

Also, there is a device-dependent rule for physical sequential data sets:

Track-overflow option for direct-access devices; when this option is used, a record that does not fit on a track is partially written on that track and continued on next track; if this option is not used, records are not split between tracks.

Figure 26. Record Formats — Physical Sequential Data Sets Without Keys (cont'd)



The same rules apply to physical sequential data sets with keys as for those without keys; also:

- All keys in data set must be the same length.
- Number of bytes transmitted in a READ or WRITE operation equals the key plus the data portion of record.

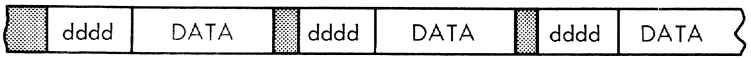
Note: Non-zero KEYLEN operand in DCB identifies data set with keys.

Figure 27. Record Formats — Physical Sequential Data Sets With Keys

Fixed-length, Blocked and Unblocked (Format F)

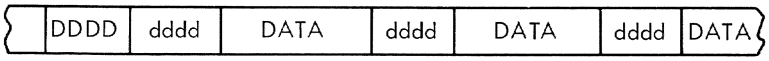
- Maximum record length - 32,760 bytes
- Buffer offset not supported
- Data in EBCDIC form is translated to ASCII

Variable-length, Unblocked (Format D)



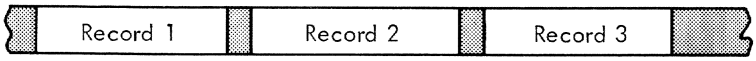
- Maximum logical record length - 32,756 bytes
- Block descriptor in example has been stepped over
- Each logical record must describe its own length; this information must be included as first four bytes of each record:
  dddd - unpacked decimal number specifying length in bytes
- dddd and DATA are translated to ASCII
- Buffer offset of 0 and 4 are supported

Variable-length Blocked (Format DB)



- Maximum logical record length - 32,763 bytes
- System performs length checking of blocks containing Format-D records, based on user supplied length information; when data sets with Format-D records (either blocked or unblocked) are created, a 4-byte control block in the form DDDD is required, where:
  DDDD - unpacked decimal number specifying block length in bytes
  Value of DDDD is determined by adding the dddd's of the records within the blocks and adding 4 bytes for the control field.
- DDDD, dddd, and DATA are translated to ASCII

Undefined (Format U)



- Maximum record length - 32,767 bytes
- Each block is treated as a logical record
- No length checking is performed
- User must make length of each Format-U record available to system in data set's data control block
- Buffer offset not supported
- Format U is supported when 128 character set is used
- Data translated to ASCII

Note: This represents the output after the system has processed the internal EBCDIC data format described in Figure 27.

Figure 28. Output Record Formats for ASCII Tapes

When more than one page number is indicated, the major reference is first. All references are within plus or minus one of the indicated page number.

IBM Time Sharing System Data Management Facilities Printed in U.S.A. GC28-2056-2

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601