An Introduction to the
NCAR Data Storage System

Jeanne Adams
Gary Aitken
Ben Domenico
Dave Fulker
Dick Valent
PREFACE

The material presented here is intended to serve as an introductory text for some of the features relating to data storage within the NCAR computer system. The sections include discussions of storage on various devices and transfers of the data in a file during program execution. It is not intended to be exhaustive; readers may need to refer to the mass storage device reference manual for details and an inclusive list of parameters and statements available to users.

The reader who is unfamiliar with the NCAR data storage system and who would like only a conceptual understanding of the system can skip the sections marked with * without loss of continuity. However, the user is encouraged to read the entire document before attempting to use the system.

The sections were written by members of a writing committee formed in an effort to provide users with an easy-to-understand document on using the new I/O. Both this manual and the reference manual will be made available to users. Whether both manuals or a combination of the two manuals will be supported is yet an unanswered question. Input from users will in the final analysis determine the most effective approach. Manuals change continuously and the changes included in any issue should improve its quality, especially if difficult text is reported by the users whom the document was written to serve.

The committee enjoyed the writing of this manual; we all learned a lot about the implementation and about "team" writing--something I had never tried before. It seems to work well and I, for one, found it a rewarding experience. The team authors are listed on the cover. Linda Besen edited the final text and she and Sara Ladd prepared the copy. Sara Ladd prepared the index.

Jeanne Adams
University Liaison and Information Services Manager
# TABLE OF CONTENTS

## CHAPTER 1

**INTRODUCTION** ........................................... 1.1  
  Processing Data ........................................ 1.1  
  TLIB Staging System .................................... 1.1  
  Data Storage System .................................. 1.1  
  Device Linkages ....................................... 1.2  
  Design Goals ......................................... 1.2  
  System Advantages .................................... 1.2  
  Chapter Contents ..................................... 1.2  
  Terminology Conventions .............................. 1.4  
  Scope of Descriptive Material ...................... 1.4

## CHAPTER 2

**DATA STRUCTURES AND ACCESS** .......................... 2.1  
  Logical Data Structure ................................ 2.1  
    Record ............................................... 2.1  
    Dataset ............................................ 2.2  
    Volume ............................................. 2.3  
  Data Access .......................................... 2.4  
    Volume Access ...................................... 2.5  
    Dataset Access ..................................... 2.6  
    Record Access ...................................... 2.6  
    + Clarification of Access Methods ................ 2.7  
    ♦ Physical Data Structure .......................... 2.9  
    ♦ Blocking .......................................... 2.9  
    ♦ Software Operation ................................. 2.10

## CHAPTER 3

**VOLUME MANAGEMENT** .................................... 3.1  
  Volume Images ....................................... 3.1  
    The *VOLUME Card ................................... 3.1  
    Volume Names ...................................... 3.2  
    Use of Keywords ................................... 3.2  
    *VOLUME Card Continuation .......................... 3.3  
  Establishing the Disk Image of a Volume .......... 3.3  
    Default Volume Establishment ...................... 3.3  
    Stage-In ............................................ 3.3  
  Retaining Volume Images .............................. 3.4  
    Automatic Volume Retention ......................... 3.4  
    Stage-Out ........................................... 3.5  
  Clarifications and Cautions ......................... 3.6  
    Default Tape Encoding ................................ 3.6  
    The NM Option ...................................... 3.6  
    Previous Versions ................................... 3.6  
    Error Recovery on MSD .............................. 3.6

† For a conceptual understanding of the DSS, these sections may be omitted without loss of continuity. Readers wishing to use the DSS are encouraged to read this entire document.
### Volumes of Tape

- Tape Drive Status
- Encoding Formats
- Segmented FORTRAN Records
- Tape Labels
- Multiple Files on a Tape
- Number of Datasets On a Volume
- Maximum Record Size
- Error Options on Tape
- Tape Length and Volume Size Limit

### Control Features

- Blocking Parameter
- Volume Protection on Disk

### Dataset Management

- Dataset Connection
- Old and New Datasets
- Open Statement
  - Unit Number
  - Dataset Name
  - New vs. Old Status
  - Access Method
  - Limits for New Datasets
  - Error Branch
  - Dataset Sequence Number
- Close Statement
  - Unit Number
  - Keep vs. Delete Status
  - Error Branch
- Inquire Statement
  - Unit Number
  - Dataset Name
  - Dataset Sequence Number
  - Dataset Specification
  - Existence
  - Access Method
  - Open Status
  - Dataset Limitations
  - Error Branch
  - Dataset Sequence Number vs. Name
  - Volume Name

† For a conceptual understanding of the DSS, these sections may be omitted without loss of continuity. Readers wishing to use the DSS are encouraged to read this entire document.
CHAPTER 5

RECORD MANAGEMENT ....... 5.1
Reading and Writing Records ....... 5.1
† Sequential Access, Unformatted ....... 5.1
† Sequential Access, Formatted ....... 5.2
† Formatted vs. Unformatted ....... 5.2
† Direct Access ....... 5.3
† Error Branch ....... 5.3
† End Marker Branch ....... 5.4

CHAPTER 6

LANGUAGE STANDARDS FOR FORTRAN ....... 6.1
Portability ....... 6.1
† Summary of FORTRAN Statements Associated with the NCAR Data Storage System ....... 6.2
† FORTRAN 66 ....... 6.2
† FORTRAN 77 ....... 6.3
† NCAR 7600 FORTRAN ....... 6.4
† CRAY-1 FORTRAN ....... 6.5

† APPENDIX
Keyword and Parameter Options for the *VOLUME Card 7.1

INDEX

† For a conceptual understanding of the DSS, these sections may be omitted without loss of continuity. Readers wishing to use the DSS are encouraged to read this entire document.
The NCAR Computing Facility has gradually acquired a wide range of interconnected devices on which users can store data for processing. The term NCAR data storage system (DSS) is used here to describe these devices and their inter-relationships. The primary objective of this document is to present a clear and concise picture of this system so that users will be able to utilize it for their particular data handling needs.

Until recently, the only permanent external storage medium at NCAR was 1/2" tape, and the only semi-permanent medium was the disk. The TLIB (Tape LIBrary) system, introduced some years ago, enables users to take advantage of the disk units as an intermediate storage device between an executing program and 1/2" tape. With TLIB, data are staged between 1/2" tape and the disk by the system. Executing programs then use the data on the disk. This staging process is under user control.

The acquisition of a mass storage device (MSD), the Ampex Terabit Memory System, has given NCAR additional data storage facilities. The existing TLIB system has been expanded to accommodate this new device. The disk units, 1/2" tape drives, and mass storage device, together with their associated hardware and software, constitute the NCAR data storage system (DSS).
1.2
Introduction

DEVICE LINKAGES
The linkage among the three DSS devices and a user program is shown schematically in the diagram on the following page. The diagram also summarizes important characteristics of each DSS device.

DESIGN GOALS
The overall goal of the NCAR data storage system is to provide for the efficient transfer of data from one storage device to another so that users may benefit from the rapid, random-access characteristics of the disks, the long-term, on-line properties of the MSD and the convenient portability of 1/2" tape.

SYSTEM ADVANTAGES
There are several advantages to this system:

- Tape drives are not held by a single job for long periods of time.
- Given sufficient disk space, tape images may remain on the disk for several different job executions. This reduces the total amount of tape input/output and the number of tape mounts.
- Data transfers to and from central memory involve fast access disks rather than tape drives, thereby reducing the total peripheral processor resources required.
- Disks are suitable for random (direct) access input/output operations, whereas 1/2" tape is fundamentally a sequential medium. Use of the disks makes it possible for data collections to be accessed efficiently in an order other than the order in which they were created.
1.3 Introduction

MSD
* on-line storage
* short access time (no operator intervention)
* semi-permanent
* random access
* off-line storage
* long access time (operator intervention always required)
* permanent
* sequential access

USER PROGRAM

DISC

on-line storage
short access time (no operator intervention)
semi-permanent
random access

MSD

on-line storage
medium access time (operator intervention seldom required)
permanent
sequential access

HALF-INCH TAPE

- off-line storage
- long access time (operator intervention always required)
- permanent
- sequential access
- portable
Chapter 2 of this document presents an overview of the entire DSS. The operation of the system is discussed on a conceptual level, focusing on the manner in which the system effects the transfer of data among the various devices. The next three chapters discuss the mechanisms for user control of the data transfers. In particular, Chapter 3 deals with data movement among the disks, the MSD and 1/2" tape. Chapters 4 and 5 describe the input/output statements which are used to control the transfer of data between the disks and an executing program. Since many users are concerned about the portability of their programs, the last chapter is a comparison between the FORTRAN 66 and FORTRAN 77 standards for input/output statements, and the NCAR extensions to those standards.

In an attempt to avoid cumbersome terminology while remaining consistent within this document, the following conventions have been adopted:

- "Data storage system" or "DSS" refers to the entire system being described in this document.
- "Mass storage device" or "MSD" refers specifically to the hardware and software of the Ampex Terabit Memory System, serial number 4. In other documents, this system is also called the TBM, TMS, and TMS-4.
- "Tape" refers to 1/2-inch magnetic tape as opposed to the 2-inch videotape associated with the MSD.
- "Record" is used to mean logical record unless otherwise specified.
- The words "dataset" and "file" are used interchangeably.
- The term "end marker" means whatever mark is used to indicate the end of a dataset.

Finally, in reading this document, one should keep two things in mind. First, our intention is to limit the scope of the document in order to provide a coherent overview of the fundamentals of the DSS. This is not an exhaustive reference manual. Second, at the time of this writing, the DSS is a moving target; the system is in a state of considerable flux, so some of the material will be outdated in a short time--perhaps before this document is published.
Because of the large amount of data handled at NCAR, it is not possible to allow users to store data on the disks permanently; there is simply not enough space. Consequently, data are constantly being moved from one device to another. This process is largely controlled by the user, and the mechanisms for controlling it are explained in the next three chapters. It is the purpose of this chapter to expose the general structure of all data as viewed by the data storage system (DSS) and to describe briefly the manner in which data are accessed.

**LOGICAL DATA STRUCTURE**

For the most part, users need only be familiar with the logical structure of the data handled by the DSS in order to use the DSS effectively. The logical structure is the structure of the data as viewed from the user program. It consists of records, which are gathered together to form datasets, which are in turn gathered to form volumes.
2.2
Data Structures and Access

Record

The basic unit of data read or written from a user program is known as a record. A record is the smallest unit of user data transferred in a single read or write operation.

- Records consist of one or more bits of information. The length of a record is often measured in computer words rather than bits. On the Control Data 7600, a word consists of 60 bits. On the Cray-1, a word consists of 64 bits.

- The bits in a record consist of only those data bits of which the user is aware; any extra information added by the system for control purposes is not considered to be part of the record.

- Records may be of different lengths.

- Records written by direct access (discussed later) have a maximum length associated with them. Once this maximum is established, a record cannot be rewritten with a length greater than this maximum.

- The bits of information in a record are arranged in serial order and are read or written as a continuous stream. In some cases (such as with formatted READ and WRITE statements) the bit stream represents a string of characters. Although certain operations interpret these data as characters, the stored form is simply a serial stream of bits.

Dataset

Records are grouped together into larger units known as datasets (files). At many installations, the dataset is the unit of data which is manipulated outside of the user program. At NCAR, however, only volumes (collections of datasets, described later) may be manipulated outside of the user program. A dataset must be "opened" from the user program before it can be used. Similarly, it should be "closed" when all processing is complete. The action of opening the dataset specifies the dataset to be used and other characteristics affecting the connection between the user program and the dataset.

- Datasets are usually named entities; the name may be from one to 17 alphanumeric characters, the first of which must be a letter. All datasets stored on the disk or on the MSD are named; those stored on tape may be unnamed. A dataset is considered unnamed if it does not contain a label field (see below). Datasets on a given tape are either all named or all unnamed.
Named datasets contain a special field, known as a header label, at their beginning. The label specifies such things as the dataset name and creation date.

A dataset contains zero or more records. If it contains zero records, it is known as a "null" or "empty" dataset.

Each dataset has an end marker of some type which separates it from the next dataset on the volume. This end marker is generated by the system when the dataset is "closed" after writing, and may be tested for in a user program if the dataset is being accessed sequentially. (See chapter 5 and Record Access in this chapter.) When an unlabeled volume (see below) is copied to tape, each end marker degenerates to what is normally referred to as a "file mark" or "end-of-file."

The largest unit of information handled by the NCAR DSS is a volume. A volume is a collection of zero or more datasets. These datasets appear on the volume in some order, separated by their end markers. The volume is the unit of information which may be moved among the DSS devices.

Volumes are named entities; the name may be from one to six alphanumeric characters, the first of which must be a letter. On a physical tape, the volume name corresponds to the name on the physical tape reel. The system makes no distinction among volumes by user number. To prevent duplication, users should obtain distinct names for volumes from the tape librarian.

If a volume consists of zero datasets, it is known as a "null" or "empty" volume; that is, it contains no data, other than possibly a label (see below).

A volume contains a special field, known as a volume label, at its beginning. The label specifies such things as the volume name, the machine used to generate the volume, the last date written, etc. Volumes stored on tape may have the label omitted, in which case they are referred to as "unlabeled" volumes.
DATA ACCESS

Data are stored in the DSS as a hierarchy of volumes, datasets, and records. The volume is the unit of data transferred among the external storage devices; its movement is controlled using the *VOLUME card (see chapter 3). The dataset provides a means of accessing a specific set of records within a volume; this access is controlled with the FORTRAN OPEN, CLOSE and INQUIRE statements (see chapter 4). The record is the unit of data transferred between the disk and the user program; its movement is controlled by the FORTRAN READ and WRITE statements (see chapter 5). The diagram below illustrates this concept.

Each of the logical levels (volume, dataset, record) is accessed in a different manner, and is discussed separately in the paragraphs which follow.
Volume Access

Volumes are moved from one element of the DSS to another by copying. As a consequence, there may be more than one image of a given volume in existence at any time.

The volume is the basic unit transferred among the elements of the DSS. It is not possible to move only a part of a volume (e.g., dataset, record).

The system keeps track of which volumes are stored on each element of the DSS. It does not, however, keep track of what data are on each image of a volume. In particular, it has no knowledge of which of several images (e.g., disk, tape, MSD) of a volume is the most recent. It is the user's responsibility to monitor and control the movement of data to maintain its integrity.

Images of volumes may be deleted from the disk or MSD. This deletion must be explicitly requested by the user with the following exception: if the system needs disk space, it may automatically delete an image of a volume from the disk. Unless specifically told not to do so by the user, the volume is first copied to the MSD so that it may be later recovered. This process, known as "age-off," may take place at any time when the volume to be deleted is not in use. Volumes are selected for age-off on the basis of length and time of last use.

The data stored on a volume are normally regarded simply as a string of bits, and are copied bit for bit when moved from one device to another. However, in copying a volume to or from tape, one may specify that the disk image be treated as 6-bit DPC characters, and the tape image be treated as 6-bit BCD, 8-bit ASCII, or 8-bit EBCDIC characters. Under these conditions, code conversions are performed during the copy process.

When volumes are copied to tape, the volume label may be copied along with the data if desired. When a labeled volume is copied from tape to disk, the label may be checked to see that the volume name agrees with that requested by the user. When an unlabeled volume is copied from tape to disk, a label is inserted by the system automatically.

When volumes are established on the disk or MSD, a limit is placed on the maximum number of datasets. This maximum may not be extended except by deleting the images and regenerating them.
2.6
Data Structures and Access

Volume Access

(continued)
The disk image of a volume may normally be accessed by only one executing program at a time. However, it is possible to mark the entire volume (not the individual datasets) "read only," thereby allowing more than one program to access the volume simultaneously. Once marked, the volume will remain as "read only" until the condition is specifically cleared or until the disk image is deleted.

Dataset Access

Only one dataset from a given volume is accessible at any one time. More than one dataset may be accessed simultaneously providing they are not from the same volume.

Datasets in volumes stored on tape often are unnamed (unlabeled). In such cases, the datasets are considered to be numbered sequentially from the start of the volume. When the volume is copied to disk, the datasets are given names of the form "NCARSYSTEMDR1nnnn" where "nnnn" is the dataset number, and labels are inserted by the system. When copying a volume from disk to tape, if the user chooses not to copy label fields, the datasets become unnamed. Null datasets are not copied. (Note that this may cause problems. For example, the first non-empty dataset on disk becomes the first dataset on tape. When subsequently recopied to disk, it is renamed the first dataset.)

Record Access

Records may be accessed either directly or sequentially. The records in any dataset may be accessed by either method. However, all records in a given dataset must be accessed in the same manner during any particular use of the dataset.

In direct access, records are numbered, and may be processed in any order by specifying the record number of the desired record. Attempts to access nonexistent records are considered errors.

In sequential access, records are processed from the beginning of the dataset to its end. No means are provided for jumping around in a dataset--one may only access the next record or the previous record. One may also back up to the beginning of a dataset and start over. In order to simulate starting in the middle, it is necessary to start at the beginning and move forward, one record at a time, to the desired position.
Without further details of the DSS implementation, it may be unclear which access method should be used for a particular dataset. The following should assist the user by describing the storage requirements and rules of usage for both access methods.

If a program is performing only read operations on a previously established dataset, there is essentially no difference between direct and sequential access except for the manner of specifying which record should be read. In fact, sequential access may be simulated under direct access by using an incremented counter. However, if write operations are involved, significant differences exist between the forms of access.

The differences result primarily from the procedures by which disk storage is allocated for datasets and volumes. In the case of direct access, space for the entire dataset is determined when the dataset is opened, and no additional space may be allocated. Under sequential access, however, disk space is allocated dynamically and depends upon the number and sizes of records written. In general, sequential access requires less disk space and requires fewer advance specifications than does direct access.

The DSS dynamic allocation of disk space that takes place under sequential access has some pitfalls for the unwary user. The basic principle is that when a record is written under sequential access, that record becomes the last record of the dataset and the volume, until another record is written. Previously existing records or datasets that were physically located beyond the new record are discarded. Disk space is either taken or released in accordance with requirements for the newly defined volume.

The DSS allocation of disk space under direct access is not dynamic and is based on information available when the dataset is opened. There are two cases: either the dataset currently exists on the disk (and is thus defined as an "old" dataset), or it is to be created (defined as a "new" dataset). If the dataset is old, the disk space it occupies (including dummy records, as explained below) cannot be changed. If the dataset is new, disk space is allocated on the basis of a maximum record length and a maximum number of records, both of which should be specified by the user when the dataset is opened. Specifically, when a dataset is opened for creation under direct access, dummy records (the maximum number, each of the maximum length) are written to the disk by the system to serve as placeholders. Once the maximum size characteristics have been established for a new dataset under direct access, they cannot be modified. Exception: As described
Clarification of Access Methods (continued) below, datasets copied to tape and subsequently recopied to disk may have modified size characteristics.

Caution must be exercised when changing methods of access to a dataset. The following table gives rules of usage that are organized by original mode of access versus current mode of access.

<table>
<thead>
<tr>
<th>ORIGINAL ACCESS METHOD</th>
<th>CURRENT ACCESS METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>Maximum sizes are those of the original specification.</td>
</tr>
<tr>
<td>Sequential</td>
<td>Sequential</td>
</tr>
<tr>
<td></td>
<td>Reading is permitted but dummy records that remain are skipped; this may affect the user's record counts. Writing is possible but inadvisable.</td>
</tr>
<tr>
<td></td>
<td>A record written becomes the last record of the dataset and of the volume; this may destroy other records and datasets.</td>
</tr>
</tbody>
</table>

In defining original access method for the above table, the following rule must be observed: any dataset copied to disk from tape is considered to have sequential original access. Hence a dataset created under direct access, copied to tape, and subsequently copied back to disk may behave differently than one that remains on disk or MSD. In particular, the size characteristics may be modified—when a dataset is copied from disk to tape, only the actual data portions of non-dummy records are copied. Thus unless the user has written the maximum number of records, each of maximum length, a dataset created under direct access, copied to tape, and recopied to disk, will have size and access characteristics differing from those of the original. The facility for tape storage of direct access datasets is under development.
In summary, the casual user can avoid problems by viewing the changing of access methods as a one-way process: a dataset created under direct access should not be used under sequential access (nor should it be stored on tape); however, a dataset created under sequential access (including tapes from outside sources) may be used under either sequential or direct access. Such use of direct access permits the reading of records in any order and permits the rewriting of specified records without destroying others, neither of which is possible under sequential access.

**PHYSICAL DATA STRUCTURE**

While the logical data structure is normally the only data structure with which users need be concerned, it is occasionally necessary to deal with parameters affecting the physical structure of data. For this reason, the physical structure of data as stored in the DSS is explained below.

**Blocking**

The DSS stores volumes as serial bit streams. Volume and header (dataset) labels, end markers, and data bits are simply made into a serial stream and stored as such. The order in which the items are stored is:

- volume label
- header label for dataset 1
- dataset 1 logical records
- end marker for dataset 1
- header label for dataset 2
- dataset 2 logical records
- end marker for dataset 2
- etc.
Except on tape, the data are stored in blocks of uniform size. This size must be a multiple of 122,800 bits (2048 words on the 7600), and the default is 8 units, or 983,040 bits (16,384 words on the 7600). In this storage scheme, no distinction is made among labels, end markers, and data. They are all packed as tightly as possible into the blocks. As a consequence, several records may be packed into the same block, and any given record may extend over more than one block. An example of the relationship between logical data and its physical storage structure is diagramed below:

![Diagram showing the relationship between logical data and physical storage structure]

Software Operation

The DSS contains software which serves as a data manager operating between the user program and the disk. In performing this function, it uses an area of computer memory (LCM on the 7600) for data storage. In reading a dataset, the blocks are read into this portion of memory by the DSS software. When a user program requests a record, it is delivered from this auxiliary memory area. A similar process occurs when writing. Because the blocks are normally much larger than the records, the number of disk accesses is substantially reduced in most cases.

Block size is under user control. The DSS software does not allow users to change block size for the image of a volume already on disk or MSD. However, the disk and MSD images may be deleted and recreated with a new block size if necessary. Block size is discussed in the following chapter.
As described in the previous chapter, the volume is the unit of data transferred among the three storage devices (disk, tape, MSD) under the NCAR data storage system (DSS). It is the purpose of this chapter to discuss the mechanisms by which volumes are transferred and by which volume characteristics are established. Since certain volume characteristics vary according to the storage device on which they appear, we shall sometimes refer to a disk image of a volume, an MSD image of a volume, or a tape image of a volume; when the meaning is clear, we shall simply refer to an image or a volume. On 1/2" tape, a volume is simply a reel of tape.

Under the DSS, a user's program can access individual records only from the disk image of a volume; thus, utilization of tape or the MSD involves volume transfers under DSS control. (A user may still connect a program directly to tape via the *ASSIGN card. For this discussion, however, the DSS is assumed to permit program access only to the disk.) When these transfers occur, the volume retains its original name. Thus, several copies (not necessarily identical) of a given volume can exist simultaneously, distinguished only by storage device. Many of the DSS volume management options arise from this circumstance and the need for user control of it.

In the NCAR Job Control Language (JCL), DSS volume management is controlled by use of the *VOLUME card and a collection of parameters in keyword form; i.e., KEYWORD=parameter. The *VOLUME card appears among the first control cards of a deck and its most basic function is to signify that access to a specified volume will be required at some point in the following program.
The *VOLUME Card
(continued)
or system activities. The simplest form is

*VOLUME,unit,VSN=vsn

where *VSN* is the name of the volume (six or fewer characters, the first of which is a letter) and *unit* is the integer unit number used for program references to this volume. If more than one volume is to be accessed, separate *VOLUME* cards with different unit numbers must be used.

Volume Names

As mentioned previously, volume names remain unchanged in transfers between storage devices; thus, a user who intends to invoke tapes must assign volume names that correspond to actual tape labels. If no tapes are involved, any name is permitted, but conflicts among different users assigning the same name are a potential problem. The system makes no distinction among volumes by user number. To avoid potential conflicts, names should be obtained from the tape librarian. The names DISK and DRUM cause the system to create "scratch" volumes on the disk which cannot be transferred or retained after job completion, but which are protected against conflicts from other users.

Use of Keywords

The various capabilities described in subsequent portions of this chapter are invoked by use of the *VOLUME* card and additional keywords in the form

*VOLUME,unit,VSN=vsn,KEYWORD1=parameter1,KEYWORD2=parameter2,...

where each keyword and its corresponding parameter specify some particular option. The *VOLUME* must begin in column 1 and *unit* must be the first parameter; however, keywords may appear in any order and all, except VSN, are optional. When a particular keyword does not appear, the DSS uses some default assumption about the corresponding capability, as will be described throughout this chapter.
Embedded blanks are not permitted on the *VOLUME card because the control card scan terminates on the first blank. If one card is insufficient to hold the desired specifications, continuation may be effected by additional cards of the form

*VOLUME, unit, VSN=ven, KEYWORD=parameter,...

where unit and ven match that of the first card.

**ESTABLISHING THE DISK IMAGE OF A VOLUME**

Default Volume Establishment

Under most circumstances, a *VOLUME card results in the presence of a disk image of the named volume. The default procedure is to use an existing image of the volume if possible (since an image may remain on the disk for some time). If no such image exists, a disk image of an empty volume is created, under the assumption that the user intends to generate a new volume during execution. An image of an empty volume consists only of the pointers and identifiers necessary to start a new volume on the disk. (The above procedure is similar to mounting a tape which may or may not contain data. However, the contents of the disk are more difficult to predict because of the age-off process.)

Stage-In

If a user desires some other procedure for creating the disk image, he may use the keyword, STAGEIN, in the form

STAGEIN=stagein

where stagein is a character string that determines the type of search used to locate the specified volume and establish a disk image. The following table describes the available options.
3.4
Volume Management

Stage-In (continued)

<table>
<thead>
<tr>
<th>STAGEIN=</th>
<th>System Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS (default)</td>
<td>Use disk image if available; if not, create an empty volume for subsequent use.</td>
</tr>
<tr>
<td>MA</td>
<td>Use disk image if available; if not, &quot;ascend&quot; image from MSD; fatal error if not available.</td>
</tr>
<tr>
<td>RT</td>
<td>Use disk image if available; if not, &quot;ascend&quot; image from MSD if available; if not, obtain image from tape; operator diagnoses fatal error if tape is not available.</td>
</tr>
<tr>
<td>ZS</td>
<td>Prior to any other activity, eliminate the disk image if it exists.</td>
</tr>
<tr>
<td>ZM</td>
<td>Prior to any other activity, eliminate the disk image and/or the MSD image if either exists.</td>
</tr>
<tr>
<td>NM</td>
<td>When used with RT, inhibit ascend from MSD; also, mark the disk image &quot;do not copy to MSD during age-off.&quot;</td>
</tr>
</tbody>
</table>

Where meaningful, stagein can be several of the above, in any order and connected by commas. For example, STAGEIN=ZM,RT effectively forces the disk image to be created from tape.

The NM option needs further explanation: although this option is associated with the STAGEIN keyword, which is primarily for initialization procedures, the NM value also affects activities that take place after job completion, as will be described below.

RETYAINING VOLUME IMAGES

Automatic Volume Retention

The default behavior of the DSS at job completion is as follows. Any non-empty volume images remaining on the disk are retained for an indefinite period. When demand dictates, disk images are deleted on a least-used, largest-size basis. Just prior to deletion, the image is transferred to the MSD. The user may control this behavior in several ways, one of which is to use the NM value with the STAGEIN keyword; this option inhibits copying the volume to the MSD at age-off. Thus, unless other mechanisms are utilized, a disk image created with the NM option eventually will be lost.
Stage-Out

Other mechanisms for user control at job completion are specified via the STAGEOUT keyword in the form

\[ \text{STAGEOUT=stageout} \]

where \textit{stageout} is a character string which determines volume disposition according to the following table.

<table>
<thead>
<tr>
<th>STAGEOUT=</th>
<th>System Procedure at Job Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD</td>
<td>Copy (or descend) the image to the MSD; mark the disk image as &quot;descended.&quot;</td>
</tr>
<tr>
<td>DT</td>
<td>Copy the image to tape; operator diagnoses error if tape is not available.</td>
</tr>
<tr>
<td>ZT</td>
<td>After other operations are complete, delete the image from the disk; unless specified otherwise, do not copy the image to the MSD.</td>
</tr>
<tr>
<td>MZ</td>
<td>After other operations are complete, delete the image from the MSD if it exists.</td>
</tr>
</tbody>
</table>

As before, meaningful combinations of the above, connected by commas, can be used. For example, STAGEOUT=DT,ZT causes the image to be written to tape and deleted from the disk.

Except for the ZT procedure, each of the above options leaves an image which may reside on the disk for some time. If it becomes necessary to delete this volume from the disk, it will be copied to the MSD at age-off unless the disk image has been marked "do not copy to MSD during age-off" by an NM option, or marked "descended" by an MD option. The "do not copy to MSD during age-off" mark remains on a disk image (and cannot be changed) until the image has been deleted; however, the "descended" mark can be erased by a subsequent job which, for example, modifies the disk image but does not descend it via the MD option.
Specifying the RT option for STAGEIN or the DT option for STAGEOUT implies the potential use of tape. Since there are many ways of encoding and formatting tape, several keywords, to be described in a subsequent section, are devoted to the control of these processes. The default specification is for binary (bit serial) encoding, without labels, on 9-channel tape, at a density of 1600 bpi. The user who wishes to encode characters or otherwise deviate from the default should refer to the section entitled Volumes of Tape.

The default procedure is to create MSD images of all tape volumes that pass through the DSS (via the RT and DT options). Thus, the DSS can eliminate some operator tape mounts by intercepting tape requests (RT) and inserting the corresponding MSD image if available. The assumption behind this procedure is invalid when tape modifications occur outside of the DSS. Thus, the NM option was devised to permit the use of tape without this automatic creation and insertion of MSD images. This option is useful whenever outside modifications of tapes (such as field data collection) is involved. The option does not preclude the specified use of MSD images.

Analogous to writing tape, creating an MSD image of a volume effectively destroys any previous MSD images of that volume. In other words, no facility exists for maintaining several versions of a volume. Thus, as with tape, considerable caution should be exercised in performing modifications to a volume.

When an error occurs while ascending an MSD image, the DSS provides no partial recovery for any portion of the data which remains intact: under the RT option, the MSD image is assumed not to exist and the tape is requested; under the MA option a fatal error is diagnosed. However, such errors should occur less frequently than with 9-channel tape.

If the STAGEIN parameter includes RT or the STAGEOUT parameter includes DT, use of tape is implied. The default selection is 9-channel tape at a density of 1600 bpi with the ring in place (read or write permitted). For other choices, the keyword,
TAPE, may be used in the form

$$\text{TAPE}=\text{dens}, \text{chan}, \text{read}$$

where \text{dens} defines tape density and \text{chan} defines the number of channels as follows:

- **TAPE=16,9** (default) \hspace{1cm} 1600 bpi, 9-channel
- **TAPE=8,9** \hspace{1cm} 800 bpi, 9-channel
- **TAPE=8,7** \hspace{1cm} 800 bpi, 7-channel
- **TAPE=5,7** \hspace{1cm} 556 bpi, 7-channel
- **TAPE=2,7** \hspace{1cm} 200 bpi, 7-channel

The optional \text{read} parameter may be used with any of the above, and has only one legal value, \text{R}, which indicates ring out (read only) status for the tape. This effectively inhibits the STAGEOUT=DT procedure but does not preclude writing the disk image or transferring a modified image to the MSD.

**Encoding Formats**

Data are generally encoded on tape in either binary (bit serial) or character form. Although bit serial is always a string of bits, the encoding of characters occurs in several formats: for 7-channel tape it is usually BCD (even parity); for 9-channel tape, it is usually ASCII or EBCDIC. Character encoding within the Control Data 7600 is in still another form termed DPC. The default behavior of the DSS is to assume, at the volume level, that all data are in bit serial format. For most applications which deal only with the NCAR DSS, this default is suitable and requires no conversions as volumes are transferred. However, if tape is used as a portable storage medium, it may be necessary to convert data from one format to another. A limited class of conversions may be performed by the DSS during volume transfers between disk and tape.

Within the volume management structure of the DSS, a very important simplifying assumption is made: the encoding format is uniform throughout the volume. If a tape has been acquired with mixed formats (a possibility even within the FORTRAN standards), the DSS does not provide conversion mechanisms. However, unless the tape is recorded with mixed parity types, conversion can be performed within the user's program if the encoded characters are interpreted as bit serial.

If specified, volume level conversion occurs during transfers between disk and tape. Therefore, the type of conversion is determined if the format associated with each of these devices is specified. This may be done via the keyword parameter

$$\text{CONV} = \text{conv}$$

where \text{conv} defines disk and tape formats according to the table on the following page.
Encoding Formats †
(continued)

<table>
<thead>
<tr>
<th>Value of conv</th>
<th>Format on Tape</th>
<th>Parity</th>
<th>Format on Disk</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN (default)</td>
<td>bit serial</td>
<td>odd</td>
<td>bit serial</td>
<td></td>
</tr>
<tr>
<td>AS</td>
<td>ASCII characters</td>
<td>odd</td>
<td>DPC characters</td>
<td>9-channel only</td>
</tr>
<tr>
<td>EB</td>
<td>EBCDIC characters</td>
<td>odd</td>
<td>DPC characters</td>
<td>9-channel only</td>
</tr>
<tr>
<td>DC</td>
<td>BCD characters</td>
<td>even</td>
<td>DPC characters</td>
<td>7-channel only</td>
</tr>
<tr>
<td>NC</td>
<td>bit serial</td>
<td>even</td>
<td>bit serial</td>
<td>7-channel only</td>
</tr>
<tr>
<td>LG</td>
<td>bit serial with segmented records</td>
<td>odd</td>
<td>bit serial with unsegmented records</td>
<td>see below</td>
</tr>
</tbody>
</table>

Notes

Segmented FORTRAN Records †

The LG option needs further discussion because it affects actions other than transfers between disk and tape. The explanation of this phenomenon requires discussion of certain record level functions and a flag which is associated with the disk image of a volume.

Historically, the NCAR FORTRAN unformatted (binary) READ and WRITE statements were implemented to work with segmented records; that is, the logical record (specified by the I/O list) was actually recorded as one or more physical records, each containing no more than 512 words (including two control words). As a result of upgrading these statements to perform direct access, the current implementation works either with segmented records, as above, or with unsegmented records (with logical and physical records being identical). However, direct access may be used only when the disk image has unsegmented records.

The factor which determines the mode of operation is a flag on the disk image of a volume. This flag is either on or off according to the following criteria:

- The LG option causes the flag to be off.
- A volume named DISK or DRUM (i.e., a scratch volume) always has the flag off.
- In all other cases, the flag is on.
- Once established, the flag remains fixed until deletion of the disk image.
- The flag is also carried with the MSD image; thus, any image ascended from the MSD has the flag set to its original value.
The effect of this flag is to control the type of records (segmented or unsegmented) accessed on the disk by FORTRAN unformatted (binary) READ and WRITE statements. If the flag is on, the implementation of the WRITE (or READ) statement causes segmenting (or unsegmenting), including the creation (or removal) of control words, to occur between the user's program and the disk; in this case the disk image contains segmented records. If the flag is off, no such activity takes place and the disk image contains unsegmented records. All other forms of I/O at NCAR ignore this flag and access unsegmented records. The flag must be off for the use of direct access.

Note that, under the LG option for the CONV keyword, the tape image is segmented but the disk image is not. In this case the segmenting (or unsegmenting), including the creation (or removal) of control words, occurs at stage-out (or stage-in) time; furthermore, these conversions take place for all records of the volume regardless of their method of generation.

Note also that the tape image of a record from the above FORTRAN statements is always segmented: either the LG option causes segmented records on tape, or the lack of the LG option causes the FORTRAN statements to generate segmented records on the disk.

The user may view this rather complicated arrangement in the following two simplified forms:

- Use the LG option only for volumes on which direct access via FORTRAN, unformatted, READ and WRITE statements is anticipated. Use the appropriate options for other cases as described in the previous section, Encoding Formats,

or

- Always use the LG option, except for reading tapes that are unsegmented (such as those generated on another computer or old tapes created with statements other than the unformatted WRITE). This option will work for all types of output statements, but the tape image will always be in the segmented, bit-serial format.
Tape Labels †

Under the default system behavior, neither the name of the volume nor any of the dataset names is recorded on the tape image of the volume. In this case, accessing the correct volume depends upon operator accuracy in selecting the right tape reel; accessing the correct dataset depends upon knowledge of its position on the tape.

However, options are available for recording and interpreting volume and dataset names as special label records on tape. This is controlled by using the keyword LABEL in the form

```
LABEL=label
```

where `label` determines the following options:

- `LABEL=LA,CL,WL` Write labels (WL) on stage-out, check labels (CL) on stage-in, and use ASCII encoding (LA) for these labels if the tape is 9-channel.
- `LABEL=LE,CL,WL` As above with EBCDIC encoding

The 2-character parameters may appear in any order and the CL and WL parameters are optional. Please note the following peculiarities:

- The parameter in `label` must include either LA or LE, but in either case labels are BCD if 7-channel tape is used.
- Absence of CL means that label records are not checked for validity; however, they are still assumed to exist. Thus a tape without labels will be misread if the LABEL keyword is used in any form.
- The user should know whether or not an existing tape has labels. If it does, the complete sequence LA,CL,WL or LE,CL,WL should probably be used. If not, the LABEL keyword should not appear at all; if it does appear, a fatal error will be diagnosed.

Multiple Files on a Tape †

Upon transferring a volume image from tape to disk (stage-in), the default behavior of the DSS is to assume the tape contains one file; i.e., transfer ceases when the first end marker is read. The user can modify this procedure via the DS keyword in the form

```
DS=ds (1≤ds≤600)
```
where \( ds \) defines the maximum number of files to be transferred at stage-in. Each file transferred becomes a named dataset on the disk image. The names are defined by the system (see the next chapter) except for labeled tapes as previously described.

Note that regardless of the value for \( ds \), transfer ceases if an end-of-tape mark or two adjacent file marks are read.

Number of Datasets On a Volume

In addition to the above function, the DS keyword affects the maximum number of datasets which a disk image may contain. If DS=\( ds \) is specified, the maximum permissible number of datasets on the disk image of the volume is the smallest multiple of 75 which is greater than or equal to \( ds \). Since the default value for \( ds \) is 1, the default limit for datasets on the disk is 75.

Upon transferring a volume image from disk to tape (stage-out), all datasets of the disk image are recorded and separated by file marks (and labels if specified). Thus the default specifies at most one file at stage-in and 75 files at stage-out.

Once a maximum number of datasets for a volume (some multiple of 75) has been established, that maximum is retained with the disk image and the MSD image. There is no provision for modifying the limit except by image deletion.

Maximum Record Size

The MS keyword determines maximum record size during stage-in from tape to disk. It is of the form

\[
MS = ms
\]

where \( 1 \leq ms \leq 32 \) and the maximum record size is \( ms \times 307,200 \) bits (\( ms \times 5120 \) words on the 7600). The default is MS=1.
3.12
Volume Management

Error Options on Tape

The DQ keyword determines the course of action if parity errors are encountered during stage-in from tape to disk. It is of the form

\[ \text{DQ} = dq \]

where \( dq \) is one of the following:

- \( dq = 0 \) (default) All records are transferred to the disk and each record with an error is flagged.
- \( dq = 1 \) Any error causes job termination.

Tape Length and Volume Size Limit

The TL keyword limits the size of a volume. Its design permits the user to be sure that a volume will fit on its corresponding reel of tape. The form is

\[ \text{TL} = tZ \]

(default 25)

where \( tZ \) is the tape length in hundreds of feet. The limit is enforced by returning an end-of-tape status when a write is issued that would occur at or beyond the specified tape length (if the volume were staged-out to tape). This check incorporates information from the *VOLUME card such as density and number of channels. If \( tZ \) is greater than 25, the check is performed against a maximum of 600 million bits (10 million words on the 7600).

CONTROL FEATURES

Blocking Parameter

The transfer of records between disk and program-addressable memory actually involves system transfers of data units called blocks. For reasons not discussed here (including the need to control LCM allocation) the user may wish to control the size of these blocks. This is done via the BK keyword in the form

\[ \text{BK} = bk \]

where \( bk \) is an integer between 1 and 80 defining the block size in units of 2048₁₀ words. The default value is \( \text{BK} = 8 \).

Once established, the block size is retained with the disk and MSD images and cannot be modified unless these images are deleted.
Volume Protection on Disk

It is possible to mark a disk image for read only by using the LOCK keyword in the form

\[ \text{LOCK} = \text{lock} \]

where \( \text{lock} \) is a character string with the following values and meanings.

- **SP**: After job completion, mark the disk image "set protect;" i.e., for read only.
- **RP**: Before job execution, mark the disk image "release protect;" i.e., for read or write.
- **RP,SP**: Permit writing during the job execution and then mark the disk image "set protect."

This property remains in effect until modified by a LOCK keyword or until the image is deleted from the disk. The property is not transferred to any other storage device. Unless specified otherwise, a disk image will be marked "release protect" upon creation.

Note that it is possible for several users to access a given volume simultaneously only if the corresponding disk image is "set protect;" i.e., marked for read only. Otherwise, a user has sole access to a volume for the entire duration of the job; other users desiring access are placed on hold until the aforementioned job is complete.
In using the DSS, executing programs write records to and read records from the disk. These records are part of a specific dataset, which in turn is part of a specific volume. The *VOLUME card establishes a disk image of the volume and associates the volume with a logical unit number; any reference to that unit in a program is actually a reference to the specific volume. Thus a unit number uniquely identifies a disk image of a volume and, for brevity, the term unit or unit number will be used in this chapter.

Before any records can be read or written, the system must be informed of the specific dataset to which the records belong; that is, a connection must be made between the program and the dataset to be read or written. The process of establishing a connection between a program and a dataset on a unit is known as opening the dataset; severing this connection is known as closing the dataset. Records may be read or written only when the dataset is open. Only one dataset on a unit may be open at any time, although datasets from several units may be open simultaneously. This chapter describes the FORTRAN OPEN statement used to establish dataset connections, the CLOSE statement used to sever dataset connections, and the INQUIRE statement used to ascertain the properties of datasets and their connections.

Old and New Datasets

If a dataset exists on the unit at the time the dataset is opened, the dataset is referred to as an "old" dataset (even if it is empty); if the dataset does not exist on the unit, it is "new" and the act of opening it creates an empty dataset.

As discussed in chapter 2, datasets may be accessed either directly or sequentially. When a dataset is opened, the type of access is established. If unspecified, sequential access is used.
Old and New Datasets
(continued)

The maximum number of records allowed in a dataset and the maximum record length are characteristics which affect the way in which the DSS stores a dataset. These characteristics are important only for datasets which will be accessed directly and must be specified whenever a new dataset is established for which direct access will be used. For old datasets, these values should be omitted, as the original values are always used.

OPEN STATEMENT

The FORTRAN OPEN statement establishes a connection between the program and a dataset on a unit. It has the general form

```
OPEN(UNIT=unit,KEYWORD1=parameter1,KEYWORD2=parameter2,...)
```

The dataset which is opened must belong to the volume currently associated with *unit*. If the dataset is new, it will be made a part of this volume. Except as noted below for the unit number (*unit*), all parameters appear in the form "KEYWORD=parameter". Parameters may appear in any order, and all except the unit number are optional. The default values for omitted parameters are described below.

Unit Number +

In the OPEN statement, the user must specify the unit number with the keyword UNIT in the form

```
UNIT=unit
```

where *unit* may be any integer expression. This is the only parameter which is required in the OPEN argument list. The unit number must be associated with a specific volume via a *VOLUME* card. In all subsequent I/O statements, this unit number identifies the dataset being used. The keyword UNIT may be omitted, in which case *unit* must occupy the first position in the argument list.
Dataset Name

The name of the dataset to which connection is desired is specified with the keyword file in the form

\[ \text{FILE} = \text{file} \]

where \( \text{file} \) is a character constant or array. Execution of the statement

\[ \text{OPEN(UNIT=unit, FILE=fiZe)} \]

causes the dataset specified by \( \text{file} \) to be opened for sequential access. (Opening for direct access requires other parameters, discussed below.)

The character string in \( \text{file} \) may contain from one to 17 characters and must begin with a letter—for example,

\[ \text{FILE} = '\text{BUGABOO23}' \]

When an array is used, it must be dimensioned for two (two 7600 words are required to hold 17 characters). For example,

\[ \text{FILE} = \text{IVAN} \]

where \( \text{IVAN}(1) = '\text{BUGABOO23}' \) and \( \text{IVAN}(2) = ' ' \). When the FILE parameter is not specified by the user, the system opens to the first dataset of the volume connected to unit \( \text{unit} \). As discussed in chapter 2, unnamed datasets on tape volumes are given default names by the system when they are copied to disk. These names are of the form 'NCARSYSTEMDRInnnn', where nnnn is the numeric position of the dataset from the start of the volume. When new datasets are being generated, similar names are automatically supplied if \( \text{file} \) is omitted.

New vs. Old Status

In the OPEN statement, the user may specify whether the dataset is new or old (i.e., whether it exists on the unit prior to the execution of the OPEN statement). This is accomplished by using the keyword STATUS in the form

\[ \text{STATUS} = \text{status} \]

where \( \text{status} \) is the appropriate character constant 'NEW' or 'OLD', or a variable containing one of these values.
New vs. Old Status

(continued)

When STATUS='NEW' is used, an error exit is taken if the dataset already exists on the unit. In this way, the user prevents datasets from being overwritten unintentionally.

When STATUS='OLD' is used, an error exit is taken if the dataset does not exist on the unit.

Two other values are permissible for the STATUS parameter: 'SCRATCH' and 'UNKNOWN'. If the user desires that a dataset be deleted from the disk when it is closed, the value 'SCRATCH' should be used. When neither deletion nor the above-mentioned checking for existence is desired, the value 'UNKNOWN' may be used. The default is 'UNKNOWN'.

Access Method

The method to be used for accessing records within the dataset may be specified via the ACCESS keyword in the form

\[ ACCESS=access \]

where \( access \) is the appropriate character constant 'DIRECT' or 'SEQUENTIAL', or a variable containing one of these values. The default value is 'SEQUENTIAL'. The details of these access methods are discussed in chapter 2.

Limits for New Datasets

When a new dataset is being opened for direct access, the user must specify the maximum length. This specification is optional for new datasets opened for sequential access. The keyword RECL in the form

\[ RECL=recl \]

is used, where \( recl \) may be any integer expression. There is no default value.

The maximum number of records may also be specified for new datasets using the keyword MAXREC in the form

\[ MAXREC=maxrec \]

where \( maxrec \) may be any integer expression. There is no default value when sequential access is used, but \( maxrec \) defaults to 1,000 for direct access.

Both parameters may be specified when the dataset is old, but they must match the original values or an error condition occurs.
Error Branch +

When an error occurs during the execution of an OPEN statement, an error message is normally printed and the program terminates. If the user desires that the program continue execution instead, the ERR keyword may be specified in the form

\[
\text{ERR=err}
\]

where \( err \) is a statement label. The program will then branch to statement \( err \) if an error occurs. (A mechanism for program determination of the type of error is currently under system development.)

Dataset Sequence Number +

When a dataset is created, the system assigns a number between 1 and 600 to it. This number reflects the order in which the datasets are generated on the volume. This number, rather than the dataset name, may be used to reference the dataset. The keyword FILESEQ is used in the form

\[
\text{FILESEQ=fileseq}
\]

where \( fileseq \) may be any integer expression.

CLOSE STATEMENT

The FORTRAN CLOSE statement is used to sever the connection between a program and a dataset on a unit. It has the same general form as the OPEN statement, that is

\[
\text{CLOSE(UNIT=unit,KEYWORD1=parameter1,KEYWORD2=parameter2,...)}
\]

This statement closes whichever dataset is currently open on \( unit \). If no dataset is open on \( unit \), the CLOSE statement does nothing. Except as noted below for the unit number, all parameters appear in the form KEYWORD=parameter. All parameters except the unit number are optional and may appear in any order. The default values for omitted parameters are noted below.
4.6 Dataset Management

**Unit Number**

The unit number associated with the dataset being closed must appear in the CLOSE argument list. It is used with the UNIT keyword in the form

\[
\text{UNIT} = \text{unit}
\]

where \( \text{unit} \) may be any integer expression. It may also appear without its keyword, in which case it must occupy the first position in the calling sequence.

**Keep vs. Delete Status**

In the CLOSE statement, the user may specify whether the dataset being closed is to be kept on the disk or deleted from it. This is indicated by using the keyword STATUS in the form

\[
\text{STATUS} = \text{status}
\]

where \( \text{status} \) is the appropriate character constant, 'KEEP' or 'DELETE;', or a variable containing one of these values. When 'DELETE' is used, deletion occurs during execution of the CLOSE statement. The default value is 'KEEP'.

**Error Branch**

When an error occurs during execution of a CLOSE statement, an error message is normally printed and the program terminates. If the user desires that the program continue execution instead, the ERR keyword may be specified in the form

\[
\text{ERR} = \text{err}
\]

where \( \text{err} \) is a statement label. Then, if an error occurs, the program will branch to statement \( \text{err} \) instead of printing the error message and terminating. (A mechanism for program determination of the type of error is currently under system development.)
INQUIRE STATEMENT

The FORTRAN INQUIRE statement is used to determine characteristics of the connection between a program and a dataset on a unit. It may also be used to determine specific dataset characteristics. It has the general form

```
INQUIRE(UNIT=unit,KEYWORD1=parameter1,KEYWORD2=parameter2,...)
```

Except as noted below for the unit number, all parameters appear in the form `KEYWORD=parameter` and may appear in any order. All parameters except the unit number are optional. The keywords `UNIT`, `FILE`, and `FILESEQ` are used in various combinations to specify a particular dataset about which information is desired. Except for `ERR` (discussed below), the remaining keywords are used to indicate what information is sought and to provide variables and arrays in which the system returns the requested information.

Unit Number ↑

The executing program refers to the disk image of a volume in terms of the unit number previously associated with it on a *VOLUME card. In the INQUIRE statement, the user must specify the unit number with the keyword `UNIT` in the form

```
UNIT=unit
```

where `unit` may be any integer expression. This is the only parameter required in the INQUIRE parameter list. The keyword `UNIT` may be omitted, in which case `unit` must occupy the first position in the parameter list.

Dataset Name ↑

A specific dataset may be selected by name using the keyword `FILE` in the form

```
FILE=file
```

where `file` is a character constant or array containing the name of the dataset. The character string in `file` may contain from 1 to 17 alphanumeric characters and must begin with a letter. When an array is used, it must be dimensioned for two. (Two 7600 words are required to store a full 17 characters.) For example:

```
FILE=JANE
```

where `JANE(1)='WEATHER5'` and `JANE(2)=' '`. 
When a dataset is created, the system assigns a number between 1 and 600 to it. This number, the dataset sequence number, reflects the order in which the datasets are generated on the volume; the user may select it, rather than the dataset name, to reference the dataset. The keyword FILESEQ is used in the form

\[ \text{FILESEQ}=\text{fileseq} \]

where \( \text{fileseq} \) may be any integer expression.

The parameters described above may be combined in several ways to specify the dataset in the INQUIRE statement. In the discussion which follows, the term, specified dataset, refers to the particular dataset selected as shown in the following table.

<table>
<thead>
<tr>
<th>DATASET IDENTIFIER</th>
<th>SPECIFIED DATASET</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT=unit</td>
<td>If a dataset is open on ( \text{unit} ), that dataset is specified. Otherwise, the first dataset on ( \text{unit} ) is specified.</td>
</tr>
<tr>
<td>UNIT=unit, FILE=file</td>
<td>The dataset named ( \text{file} ), on ( \text{unit} ), is specified.</td>
</tr>
<tr>
<td>UNIT=unit, FILESEQ=fileseq</td>
<td>The dataset with sequence number ( \text{fileseq} ), on ( \text{unit} ), is specified.</td>
</tr>
<tr>
<td>UNIT=unit, FILE=file, FILESEQ=fileseq</td>
<td>The use of this combination of parameters is discouraged. Under normal conditions, it is equivalent to UNIT=unit, FILESEQ=( \text{fileseq} ). However, in some cases, the dataset specification may be ambiguous.</td>
</tr>
</tbody>
</table>
Existence †

The user may inquire whether the specified dataset exists on the disk image of the volume associated with the given unit number, that is, whether the dataset is old or not. To do this, the EXIST keyword is used in the form

\[ \text{EXIST}=\text{exist} \]

where \( \text{exist} \) must be a logical variable. It is returned with the value .TRUE. if the dataset exists on \( \text{unit} \); otherwise, it is returned with the value .FALSE..

Access Method †

When the specified dataset is open, the user may determine the access method being used with it. To do this, the keyword ACCESS is used in the form

\[ \text{ACCESS}=\text{access} \]

where \( \text{access} \) may be any variable. The appropriate character string, 'DIRECT' or 'SEQUENTIAL', is returned. When the specified dataset is not open, the value of ACCESS is undefined.

Open Status †

The INQUIRE statement may be used to determine whether the specified dataset is open (i.e., connected) with respect to the given unit number. To do this, the user specifies the dataset in the argument list, together with the keyword OPENED in the form

\[ \text{OPENED}=\text{opened} \]

where \( \text{opened} \) is a logical variable. The value of OPENED is returned as follows:

- if the specified dataset exists on \( \text{unit} \) and is open, .TRUE. is returned
- if the specified dataset exists on \( \text{unit} \) but is not open, .FALSE. is returned
- if the specified dataset does not exist on \( \text{unit} \), the value of OPENED is undefined.
4.10
Dataset Management

Dataset Limitations + The number of records on the specified dataset may be determined using the keyword NUMREC in the form

\[
\text{NUMREC} = \text{numrec}
\]

where \( \text{numrec} \) is an integer variable. The value returned in \( \text{numrec} \) is the number of records actually written on the specified dataset; if the dataset was established using direct access, it includes empty records. If the dataset does not exist, \( \text{numrec} \) is undefined.

The maximum number of records in the dataset may also be obtained if it has been established; that is, if it was explicitly specified in the OPEN statement at creation time and the dataset has never been staged in from tape (see Clarification of Access Methods, chapter 2). To obtain this value, the user specifies the keyword MAXREC in the form

\[
\text{MAXREC} = \text{maxrec}
\]

where \( \text{maxrec} \) is an integer variable. The value of \( \text{maxrec} \) is returned as follows:

- if the dataset exists on \( \text{unit} \), the maximum number of records is returned if it has been established
- if the dataset exists on \( \text{unit} \) but the maximum number of records has not been established, zero is returned
- if the dataset does not exist on \( \text{unit} \), \( \text{maxrec} \) is undefined.

If the dataset has ever been opened for direct access, \( \text{maxrec} \) is always equal to \( \text{numrec} \).

The maximum record length in the dataset may also be obtained if it has been established; that is, if it has been explicitly specified in the OPEN statement at creation time and the dataset has never been staged in from tape (see Clarification of Access Methods, chapter 2). To obtain this value, the user specifies the keyword RECL in the form

\[
\text{RECL} = \text{recl}
\]

where \( \text{recl} \) is an integer variable. The value of \( \text{recl} \) is returned as follows:

- if the dataset exists on \( \text{unit} \), the maximum record length is returned if it has been established
- if the dataset exists on \( \text{unit} \) but the maximum record length has not been established, zero is returned
- if the dataset does not exist on \( \text{unit} \), \( \text{recl} \) is undefined.
When an error occurs during the execution of an INQUIRE statement, an error message is normally printed and the program terminated. If the user desires the program to continue execution instead, the keyword ERR may be specified in the form

\[ \text{ERR}=\text{err} \]

where \( \text{err} \) is a statement label. The program will then branch to statement \( \text{err} \) if an error occurs, instead of printing a message and terminating. (A mechanism for program determination of the type of the error is currently under system development.

When the user knows the name of a dataset and wishes to know its sequence number, the SEQFIL keyword may be used in the form

\[ \text{SEQFIL}=\text{seqfil} \]

where \( \text{seqfil} \) is an integer variable. The statement

\[ \text{INQUIRE (UNIT}=\text{unit},\text{FILE}=\text{file},\text{SEQFIL}=\text{seqfil}) \]

returns the dataset sequence number in \( \text{seqfil} \) when the dataset specified by \( \text{file} \) exists on \( \text{unit} \); otherwise, \( \text{seqfil} \) is undefined.

Conversely, the user may determine the dataset name if the sequence number is known. To do this, the keyword NAME is used in the form

\[ \text{NAME}=\text{name} \]

where \( \text{name} \) is an array dimensioned for two. The statement

\[ \text{INQUIRE (UNIT}=\text{unit},\text{FILESEQ}=\text{fileseq},\text{NAME}=\text{name}) \]

returns the dataset name in \( \text{name} \) when the dataset specified by \( \text{fileseq} \) exists on \( \text{unit} \); otherwise, \( \text{name} \) is undefined.

The name of the volume whose disk image is associated with the given unit number may be obtained using the keyword VSN in the form

\[ \text{VSN}=\text{vsn} \]

where \( \text{vsn} \) is a variable in which the volume name is returned.
READING AND WRITING RECORDS

Once a unit number is associated with a volume (*VOLUME card, chapter 3), and once a dataset in the volume is open (OPEN statement, chapter 4), records in the dataset may be read or written. Records are read using the READ statement and written using the WRITE statement. The general form of these statements is:

```
READ (UNIT=unit, KEYWORD1=parameter1, KEYWORD2=parameter2,...) list
WRITE(UNIT=unit, KEYWORD1=parameter1, KEYWORD2=parameter2,...) list
```

Except as noted below for the unit number and format, all parameters appear in the form KEYWORD=parameter and may appear in any order. All except the unit number are optional. The default values for omitted parameters are noted below.

As discussed in chapter 2, records may be accessed in one of two ways, directly or sequentially. When reading and writing records, the access method used must match the one specified in the OPEN statement.

Sequential Access, Unformatted

The following form may be used for unformatted sequential access READ and WRITE statements:

```
READ (UNIT=unit) list
WRITE(UNIT=unit) list
```

where unit is an integer expression specifying the unit number and list is a list of variable, array element, and array names. (Implied DO loops are permitted.) In all READ or WRITE statements, the keyword UNIT may be omitted, in which case unit must be the first argument in the parameter list. The effect of an unformatted WRITE statement is to cause the data in the list elements to be transferred as a serial bit stream to the disk image of the dataset. These data are grouped as a single
5.2 Record Management

logarithm record. (See chapter 3 for a discussion of physical record segmentation.) The READ statement transfers data in the opposite direction. The list in a READ statement need not be of the same length as the record. If the list is shorter, the last portion of the record is ignored. If the list is longer, the additional list elements are undefined.

Sequential Access, Formatted †

If formatted reading or writing using sequential access is desired, the FMT keyword is used as follows:

`READ (UNIT=unit,FMT=fmt)list`

`WRITE(UNIT=unit,FMT=fmt)list`

where \( fmt \) is the format statement number. The FMT keyword may also be omitted, in which case \( fmt \) must be the second parameter in the argument list.

The effect of a formatted WRITE statement is to cause the list elements to be converted into a string of DPC characters via the format specifications. These characters are treated as a serial bit stream and written to the disk image of the dataset. The characters are grouped into one or more records depending on the format statement specifications. (See the FORTRAN manual for a discussion of formats.) The READ statement transfers data in the opposite direction. It should be noted that the FORMAT statement determines the number of records transferred by a READ or WRITE statement.

Formatted vs. Unformatted †

It is possible to achieve portable I/O via formatted READ and WRITE statements. Under the DSS, this requires formatting all records of all datasets on a volume (so that the disk image is entirely in DPC) and converting to a standard character code during stage-out to tape. (See Encoding Formats, chapter 3, for a discussion of conversion options.) Although unformatted I/O is, in general, nonportable due to differences in machine word structure, it is faster.
Direct Access †

If a dataset is open for direct access, the record to be read or written must be specified using the keyword REC in the form

\[ \text{REC}=\text{rec} \]

where \( \text{rec} \) is an integer expression whose value is the record number. The presence of this parameter indicates direct access; it may not appear in a sequential access statement. Records read or written under direct access may be either formatted or unformatted. The user should be aware of the following points:

- Correct usage of unformatted direct access READ and WRITE statements requires that CONV=LG be specified on the *VOLUME card unless the volume name is DISK or DRUM (see chapter 3).
- Since a formatted READ or WRITE statement may transfer several records, direct access using format conversion requires careful use of the record number. In this case, \( \text{rec} \) refers to the first record to be transferred.

Error Branch †

When an error occurs during the execution of a READ or WRITE statement, an error message is normally printed and the program terminates. If the user desires the program to continue execution instead, the keyword ERR may be specified in the form

\[ \text{ERR}=\text{err} \]

where \( \text{err} \) is a statement label. The program will then branch to statement \( \text{err} \) if an error occurs, instead of printing a message and terminating. (A mechanism for program determination of this type of error is currently under system development.)
When the user desires to detect the end marker of a dataset being read sequentially, the keyword END may be specified in the form

\[ \text{END=}	ext{end} \]

where \text{end} is a statement label. This parameter may only appear in a sequential access READ statement. If an end marker is encountered during such a READ operation, the program will branch to statement \text{end}.

The following unusual behavior should be noted:

- Whenever the DSS is being used (i.e., a *VOLUME card is used to associate volumes with logical unit numbers), one may read past the end marker of the dataset currently open (regardless of whether the keyword END is present or not). This effectively closes the original dataset and opens the next one on the volume to sequential access. This practice of reading past the end marker of a dataset is strongly discouraged.

- When the last dataset on a volume is reached, the \text{end} branch will be taken for all subsequent read attempts. The program will not terminate; if in a loop, it will run until the time limit is exceeded.
Users often encounter difficulty in transporting programs from machine to machine and from one computing facility to another. A bewildering variety of so-called "Fortrans" exist. A particular FORTRAN supplied by a computer manufacturer may become a de facto standard for FORTRAN if the compiler is widely distributed. However, there has been a national standard for FORTRAN published by the American National Standards Institute since 1966 that most programmers follow when planning a program for use on several computers. It is often referred to as the "66 Standard."

In the past five years or more, a technical committee has been working on a FORTRAN standard to replace the 1966 standard for FORTRAN. Its work is essentially finished and is known as FORTRAN 77. Among other things, it has resulted in new syntax for input/output statements. NCAR FORTRAN has many of the new features of FORTRAN 77. However, in order to provide all the necessary functions associated with the mass storage device, a number of extensions to FORTRAN 77 have been added.

So that programmers may write programs that conform to one of the above FORTRANS, a table of input/output syntax for each is included on the following pages.

**Portability**

Portability of FORTRAN programs that access datasets involves both FORTRAN language concepts and the attributes of the dataset itself. The FORTRAN statements listed indicate that a particular statement will work on another computer that supports that standard but it does not guarantee that the user's dataset will have attributes that lend it to easy transfer. Input/output statements have traditionally been the most difficult statement to port.
The statements in the table are divided into Input/Output Statements, Auxiliary Statements, and File Positioning Statements. Square brackets indicate optional items in the syntax.

**SUMMARY OF FORTRAN STATEMENTS ASSOCIATED WITH THE NCAR DATA STORAGE SYSTEM**

**FORTRAN 66**

**INPUT/OUTPUT STATEMENTS**

- `READ (u[,f])[io-list]`
- `WRITE (u[,f])[io-list]`

**AUXILIARY STATEMENTS**

- `none`

**FILE POSITIONING STATEMENTS**

- `REWIND u`
- `BACKSPACE u`
- `ENDFILE u`
FORTRAN 77

**INPUT/OUTPUT STATEMENTS**

READ ([UNIT=]u[, [FMT=]f][,REC=rm][,IOMAT=ios][,ERR=s] [,END=s]) [ioist]

WRITE ([UNIT=]u[, [FMT=]f][,REC=rm][,IOMAT=ios][,ERR=s] [,END=s]) [ioist]

**AUXILIARY STATEMENTS**

OPEN ([UNIT=]u[,IOMAT=ios][,ERR=s][,FILE=fin][,STATUS=sta] [,ACCESS=acc][,FORM=fm][,RECL=r][,BLANK=blank])

CLOSE ([UNIT=]u[,IOMAT=ios][,ERR=s][,STATUS=sta])

INQUIRE ([UNIT=]u[,IOMAT=ios][,ERR=s][,EXIST=ex][,OPENED=od] [,NUMBER=nm][,NAMED=nm][,NAME=fn][,ACCESS=acc] [,SEQUENTIAL=seq][,DIRECT=dir][,FORM=fm][,FORMATTED=fmt] [,UNFORMATTED=unf][,RECL=rl][,NEXTREC=nr][,BLANK=blank])

**FILE POSITIONING STATEMENTS**

BACKSPACE u

BACKSPACE ([UNIT=]u[,IOMAT=ios][,ERR=s])

ENDFILE u

ENDFILE ([UNIT=]u[,IOMAT=ios][,ERR=s])

REWIND u

REWIND ([UNIT=]u[,IOMAT=ios][,ERR=s])
NCAR 7600
FORTAN *

INPUT/OUTPUT STATEMENTS

READ ([UNIT=]u[, [FMT=]f][,REC=rn][,ERR=s][,END=s][,IOSTAT=ios])
[list]

WRITE ([UNIT=]u[, [FMT=]f][,REC=rn][,MODE=md][,ERR=s]
[, IOSTAT=ios])[list]

READ ([UNIT=]u[, REC=rn], NWORDS=n[, NSTATE=ns]) fwa

WRITE ([UNIT=]u[, REC=rn][, MODE=md], NWORDS=n[, NSTATE=ns]) fwa

AUXILIARY STATEMENTS

OPEN ([UNIT=]u[, FILE=dsi][, FILESEQ=n][, FORM=fm][, STATUS=sta]
[, RECL=rilen][, MAXREC=maxr][, ACCESS=acc][, GEN=gen][, EXPD=exp]
[, ERR=s][, PASS=pass][, IOSTAT=ios][, BLANK=blank])

CLOSE ([UNIT=]u[, STATUS=sta][, ERR=s][, IOSTAT=ios])

INQUIRE ([UNIT=]u[, FILE=dsi][, FILESEQ=n][, ERR=s][, IOSTAT=ios]
[, EXIST=ex][, OPENED=od][, NAMED=nm][, NUMBER=nu][, NAME=fn]
[, RECL=rilen][, MAXREC=maxr][, ACCESS=acc][, SEQUENTIAL=seq]
[, DIRECT=dir][, FORM=fm][, FORMATTED=fmt][, UNFORMATTED= unf]
[, NSTATE=ns][, MODE=md][, NEXTRC=nr][, SEQFIL=n][, GEN=gen]
[, EXPDT=exp][, VSN=name][, NUMREC=nr][, CRECL=cl][BLANK=blank])

SETPASS ([UNIT=]u,PASS=pass[, RWPASS=rwpas][, RPASS=rpas]
[, WPASS=wpass][, CPASS=cpas])

FILE POSITIONING STATEMENTS

BACKSPACE u

BACKSPACE ([UNIT=]u[, IOSTAT=ios][, ERR=s])

ENDFILE u

ENDFILE ([UNIT=]u[, IOSTAT=ios][, ERR=s])

* Parameters not discussed in this edition appear in the document covering the complete syntax. Other nonstandard syntax such as SKIPFILE has been omitted.
REMOVED
### Keywords and Parameters

<table>
<thead>
<tr>
<th>Keywords and Parameters</th>
<th>Parameter Values and Limits</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>integer between 1 and 3552</td>
<td>must be specified</td>
</tr>
<tr>
<td>BK=bk</td>
<td>integer between 1 and 80</td>
<td>8</td>
</tr>
<tr>
<td>CONV=conv</td>
<td>BN, DC, NC, AS, EB, or TB</td>
<td>BN</td>
</tr>
<tr>
<td>DQ=dq</td>
<td>0 or 1</td>
<td>1</td>
</tr>
<tr>
<td>DS=ds</td>
<td>integer between 1 and 600</td>
<td>1</td>
</tr>
<tr>
<td>LABEL=label</td>
<td>NL, LA, LE, CL, or WL</td>
<td>LA</td>
</tr>
<tr>
<td>LOCK=lock</td>
<td>SP or RP</td>
<td>RP</td>
</tr>
<tr>
<td>MS=ms</td>
<td>integer between 1 and 32</td>
<td>1</td>
</tr>
<tr>
<td>+ MVN=mvn</td>
<td>TM+4 characters</td>
<td>none</td>
</tr>
<tr>
<td>+ PASSWORD=password</td>
<td>8 characters</td>
<td>none</td>
</tr>
<tr>
<td>STAGEIN=stagein</td>
<td>RT, MA, NS, ZS, ZM, NM</td>
<td>NS</td>
</tr>
<tr>
<td>STAGEOUT=stageout</td>
<td>DT, MD, ZT, MZ</td>
<td>MD</td>
</tr>
<tr>
<td>TAPE=dens,ch,read</td>
<td>dens may be 16, 8, 5, or 2; ch may be 7 or 9; read may be R</td>
<td>16,9</td>
</tr>
<tr>
<td>+ SELECT=select</td>
<td>17 characters</td>
<td>none</td>
</tr>
<tr>
<td>TL=tl</td>
<td>positive integer</td>
<td>25</td>
</tr>
<tr>
<td>VSN=vsn</td>
<td>6 characters</td>
<td>must be specified</td>
</tr>
</tbody>
</table>

+ Parameters not discussed in this edition appear in the document covering the complete syntax.
INDEX

A
ACCESS=, 4.4
Access method
clarification of, 2.7
current mode, 2.8
direct, 2.7-2.9, 3.8, 5.3
in INQUIRE statement, 4.9
in OPEN statement, 4.4
original mode, 2.8
sequential, 2.7-2.9
formatted, 5.2
unformatted, 5.1
Access of data, 2.4
Access of data structures, 2.1
dataset, 2.6
record, 2.6
volume, 2.5
Age-off, system, 2.4-2.5, 3.4
Allocation
of disk space (dynamic), 2.7
of LCM, 3.12
procedures, 2.7
AS, 3.8
Ascend, 3.4
ASCII, 3.8
*ASSIGN card, 3.1

B
BCD, 3.
Binary Read and Write statement, see Statement
Bit serial, 3.8
BK=, 3.12
Block size, 3.10, 3.12
Blocking, 2.9
parameter, 3.12
blocks, data, 2.10
BN, 3.8

C
CHAN, 3.7
Channels, tape, 3.7
Chapter contents in this manual, 1.4
Character conversion, 2.5, 3.7
CL, 3.10
CLOSE statement, see Statement
CONV=, 3.9
Conversion
character, 2.5, 3.7
format, 3.7
Current mode of access, see Access method

D
Data
access, 2.4
blocks, 2.10
structures and access, 2.1
Data structures
logical, 2.1
physical, 2.9
Data Storage System (DSS), 1.1, 2.1
advantages, 1.2
design goals, 1.2
device linkages at NCAR, 1.2
introduction to, 1.1
processing data at NCAR, 1.1
Dataset (file), 2.1-2.2, 2.6
access, 2.6
and record, 2.4
connection, 4.1
limitations, 4.10
management, 4.1
maximum number of, 3.11
name, 4.3, 4.7
named, 2.3, 3.11
new, 2.7, 4.1
null or empty, 2.3
number on a volume, 3.11
D

Dataset (file) (continued)

old, 2.7, 4.1
sequence number, 4.5, 4.8, 4.11
specification, 4.8
unnamed (unlabeled), 2.6
DC, 3.8
Default tape encoding, 3.6
DEfault volume establishment, 3.3
DELETE status, 4.6
Deletion
disk image, 3.4
volume, 2.5
DENSE, 3.7
Descend, 3.5
DIRECT, 4.4
Direct access, see Access methods
Disk, 2.1, 2.4, 3.2
dynamic allocation of space, 2.7
format, 3.8
image deletion, 3.4
image of volume, 2.5-2.6, 3.1
volume establishment on, 3.3
volume named, 3.8
volume protection on, 3.13
DPC, 3.7-3.8
DQ=, 3.12
Drum, 3.2
volume named, 3.8
DS=, 3.10
DT, 3.5

F

File, see Dataset
FILE=, 4.3
File mark, see End marker
FILESEQ=, 4.5, 4.8
Flag, segmenting, 3.8
FMT=, 5.2
Format
conversion, 3.7
disk, 3.8
encoding, 3.7
mixed, 3.7
tape, 3.8
Formatted sequential access, see Access methods
FORTRAN
CRAY-1, 6.5
language standards for, 6.1
NCAR 7600, 6.4
66, 6.2
77, 6.3

G

H

I

INQUIRE statement, see Statement

J

Job Control Language, 3.1
Job termination, 3.5

K

KEEP status, 4.6
Keywords, 3.9
use of, 3.2

L

LA, 3.10
Label
record, 3.10
tape, 3.10
volume, 2.3, 2.5
LCM, allocation of 3.12

E

EB, 3.8
EBCDIC, 3.8
Empty dataset, 2.3
Empty volume, 2.3, 3.3
Encoding, default tape, 3.6
Encoding formats, 3.7
END=, 5.4
End-of-file mark, 2.3, 3.10
End marker, 2.3, 3.10
End marker branch, 5.4
ERR=, 4.11, 5.3
Error branch, 4.5-4.6, 4.11, 5.3
Errors
MSD, 3.6
parity, 3.12
tape, 3.12
EXIST=, 4.9
Index, DSS

L (continued)

LE, 3.10
Length
  maximum record, 2.7, 4.10
  record, 2.2
  tape, 3.12
LG, 3.8
Limits, 4.4
  dataset, 4.10
  volume size, 3.12
LOCK=, 3.13
Logical data structure, 2.1
Logical record, 3.8

M

MA, 3.4
Management
  dataset, 4.1
  record, 5.1
  volume, 3.1
Mark, end-of-file, see End-of-file mark
Marker, end, 2.3, 3.10
  branch, 5.4
Maximum number of datasets, 3.11
Maximum number of records, 2.7, 4.10
Maximum record length, 2.7, 3.11, 4.10
MAXREC=, 4.4, 4.10
MD, 3.5
Method, access, see Access methods
  Mixed formats, 3.7
  Mixed parity, 3.7
  MS=, 3.11
  MSD, 2.4
    error recovery, 3.6
      image of volume, 2.5, 3.1
  Multiple files on tape, 3.10
  MZ, 3.5

N

NAME=, 4.11
Name
  dataset, 4.3, 4.7
  volume, 2.3, 3.2, 4.11
Named dataset, 2.3, 3.11
NC, 3.8
NEW, 4.3-4.4
New dataset, 2.7, 4.1
NM, 3.4-3.6
NS, 3.4
Null dataset, 2.3
Null volume, 2.3
Number
  dataset sequence, 4.5, 4.8, 4.11
  maximum, of datasets, 3.11
  maximum, of records, 2.7, 4.10
  of datasets on volume, 3.11
  of records, 4.10
  unit, 3.2, 4.2, 4.6, 5.1
NUMREC= 4.10

O

OLD, 4.3-4.4
Old dataset, 2.7, 4.1
OPEN keywords, 4.2
OPEN statement, see Statement
OPEN status, 4.9
OPENED=, 4.9
Original access method, see Access methods

P

Parity, 3.8
  errors, 3.12
  mixed, 3.7
Physical data structure, 2.9
Physical records, 3.8
Physical storage structure, 2.10
Portability, 6.1
Previous versions, 3.6
Processing data at NCAR, 1.1
Protection, volume, on disk, 3.13
  release protect, 3.13
  set protect, 3.13

Q

R

Read only, ring out, 2.6, 3.7, 3.13
READ statement, see Statement
REC=, 5.3
RECL=, 4.4, 4.10
R (continued)

Record, 2.1, 2.2, 5.3
  access, 4.4, 4.10
  dataset and, 2.4
  label, 3.10
  length, 2.2, 3.11
  logical, 3.8
  management, 5.1
  maximum size, 2.7, 3.11
  number of, 2.7, 4.10
  physical, 3.8
  reading and writing, 5.1
  segmented, 3.8-3.9
  unsegmented, 3.8-3.9

Reel, tape, 3.1, 3.10
Release protect, 3.13
Retention, automatic volume, 3.4
Ring in, 3.6
Ring out, 3.7
RP, 3.13
RT, 3.4

S

SCRATCH, 4.4
Scratch volumes, 3.2
Segmented records, see Record
Segmenting flag, 3.8
SEOFIL=, 4.11
Sequence number, dataset, 4.5, 4.8, 4.11
SEQUENTIAL, 4.4
Sequential access, see Access method
Set protect, 3.13
Size
  block, 2.10, 3.12
  maximum record, 3.11
  volume, limit, 3.12
SP, 3.13
STAGEIN=, 3.3
STAGEOUT=, 3.5
Staging system, TLIB, 1.1
Standards, FORTRAN language, 6.1
Statement
  Binary (unformatted) Read and Write, 3.8
  CLOSE, 2.4, 4.5
  INQUIRE, 2.4, 4.7
  OPEN, 2.4, 4.2
  READ, 2.4, 5.1
  WRITE, 2.4, 5.1

Status, tape drive, 3.6
STATUS=, 4.3, 4.6
Storage
  disk, see Disk
  on tape, volume, 2.6
  structure, physical, 2.10
Structure
  logical data, 2.1
  physical data, 2.9
  physical storage, 2.10
System age-off, 2.4

T

Tape
  errors, 3.12
  format, 3.8
  image of volumes, 2.5-2.6, 3.1, 3.6
  labels, 3.10
  length, 3.12
  multiple files on, 3.10
  reel of, 3.1, 3.10
  TAPE=, 3.7
  Tape drive status, 3.6
  Tape encoding, default, 3.6
Termination, job, 3.5
Terminology conventions in this manual, 1.4
TL=, 3.12
TLIB staging system, 1.1

U

Unformatted (binary) Read and Write statement, see Statement
Unformatted, sequential access, see Access methods
UNIT, 3.2
UNIT=, 4.], 4.6
Unit number, 3.2, 4.2, 4.6-4.7, 5.1
UNKNOWN, 4.4
Unlabeled datasets, 2.6
Unsegmented record, see Record
Index, DSS

V

Volume, 2.1, 2.3, 2.5-2.6
  access, 2.5
  deletion, 2.5
  disposition, 3.4
  hierarchy of, 2.4
  label, 2.3, 2.5
  management, 3.1
  name, 2.3, 3.2, 4.11
  named disk or drum, 3.8
  null or empty, 2.3, 3.3
  number of datasets on, 3.11
  protection on disk, 3.13
  scratch, 3.2
  size limit, 3.12
*VOLUME card, 3.1
Volume image, 2.5, 3.1
  on disk, 2.5-2.6, 3.1, 3.3
  on MSD, 2.5, 3.1
  on tape, 2.5-2.6, 3.1, 3.6
  retaining, 3.4
VSN=, 3.2, 4.11

W

WL, 3.10
WRITE statement, see Statement

X

Y

Z

ZM, 3.4
ZS, 3.4
ZT, 3.5