SPERRY RAND

UNIVAC

FUNDAMENTALS OF COBOL

MASS STORAGE

PROGRAMMERS REFERENCE

UP-7503.4

PREFACE

This manual is another in the series of manuals entitled ‘‘Fundamentals of COBOL.’’ As with the other volumes of the series, it does not represent the COBOL implementation for any particular computer system; rather, it is intended as a basic reference source to acquaint the reader with the COBOL Random Access feature. The information in this manual is in accordance with the Random Access module, Level 2 of *USA Standard COBOL, X3.23–1968*.

The Random Access module provides the capability for accessing records of a mass storage file in a random manner according to a programmer-supplied key. This module also provides for the specification of rerun points and the sharing of memory among files.

The purpose of this manual is to introduce the concept of mass storage files and to describe those specific features of the COBOL language used to access mass storage files.

CONTENTS

PREFACE	1 to 1
CONTENTS	1 to 1
1. INTRODUCTION	1–1 to 1–1
1.1. GENERAL	1–1
1.2. BASIC TERMINOLOGY	1–1
2. FILE HANDLING	2–1 to 2–4
2.1. TECHNIQUES	2–1
2.1.1. Sequential Access/Sequential Processing	2–1
2.1.2. Random Access/Sequential Processing	2–2
2.2. FILE ORGANIZATION	2–2
2.2.1. Sequential Data Organization	2–3
2.2.2. Relative Data Organization	2–3
2.2.3. Direct Data Organization	2–4
3. PROGRAMMING CONSIDERATIONS	3–1 to 3–13
3.1. GENERAL	3–1
3.2. ENVIRONMENT DIVISION	3–1
3.2.1. FILE-CONTROL	3–1
3.2.1.1. SELECT	3–2
3.2.1.2. ASSIGN	3–2
3.2.1.3. FILE-LIMIT	3–2
3.2.1.4. ACCESS MODE	3–3
3.2.1.5. PROCESSING MODE	3–3
3.2.1.6. ACTUAL KEY	3–3
3.2.2. I-O-CONTROL	3–4
3.2.2.1. RERUN	3–4
3.2.2.2. SAME	3–5
3.3. DATA DIVISION	3–6
3.3.1. File Description	3–6
3.4. PROCEDURE DIVISION	3–6
3.4.1. DECLARATIVES	3–7
3.4.1.1. USE	3–7
3.4.2. Verbs	3–9
3.4.2.1. CLOSE	3–9
3.4.2.2. OPEN	3–10
3.4.2.3. READ	3–11
3.4.2.4. SEEK	3–12
3.4.2.5. WRITE	3–12
INDEX	1 to 3
FIGURES	
2–1. Relative File Structure	2–3
2–2. Direct File Structure	2–4

1. INTRODUCTION

1.1. GENERAL

The terms "mass storage," "random access," and "direct access" refer to storage devices capable of accessing data directly by reference to a physical location on the device, as opposed to devices such as magnetic tape which require time to scan sequentially-located records until the desired record is reached.

The Mass Storage feature (Random Access module) of COBOL is provided to permit utilization of the nonsequential file organization and random access capability of the direct access device.

The capability of processing sequentially organized files stored on non-mass storage devices is provided by the COBOL language as described in the *Fundamentals of COBOL-Language*. Since direct access devices can also be used as sequential access devices, reference is made to this manual where applicable.

1.2. BASIC TERMINOLOGY

The key terms used in this manual are defined here.

- Sequential Access

  An access mode in which a logical record read from or written to a file has both a logical predecessor and logical successor. The first access to a file accesses a record that has no logical predecessor; each successive access refers to the logical successor of the previously accessed logical record. The predecessor/successor relationships of a record are established when the record is written to a file.

  When stored on a direct access device, the logical predecessor and successor to a logical record need not be physically contiguous; however, the nature of non-mass storage devices requires that they be physically contiguous.

- Random Access

  An access mode in which specific logical records are obtained from or placed in a mass storage file in a nonsequential manner in accordance with a programmer-supplied key that is updated by the programmer as required.

- Sequential (Synchronous) Processing

  The manner of processing logical records in the order in which the records are made available. Execution of each statement must be completed before a subsequent statement can be initiated.

- Random (Asynchronous) Processing

  The manner of processing logical records where statements are not necessarily executed or completed in the order in which they were initiated.

  This processing mode is not a part of USA Standard COBOL; the definition is given here for reference only.

2. FILE HANDLING

2.1. TECHNIQUES

The COBOL Mass Storage feature provides two techniques for handling files:

- Sequential access with sequential processing
- Random access with sequential processing

2.1.1. Sequential Access/Sequential Processing

When sequential access with sequential processing is used, the logical records of a file are accessed sequentially in the order in which they were created and processed in that same order (sequentially). This technique is implemented by the COBOL language primarily for use with tape, printer, and card devices, but may also be used with mass storage devices. However, there is a substantial difference between file processing on non-mass storage and mass storage devices.

During processing, a magnetic tape file is either an input file or an output file; it cannot be both at the same time. After a record is read from tape, the reel is automatically in proper position for the next sequential READ. Any writing on tape that may occur before the reading of the next record can only be done on another (output) file. The contents of an input file remain unchanged by a READ.

In contrast, a mass storage file may be used for both input and output. A read operation may be performed on the same physical file as a write operation. The usual technique for updating a mass storage file is to read a record, process it, and then overwrite the original record with the updated version.

The actual location of a specific mass storage record is specified by an actual key similar to the ACTUAL KEY clause used in random access/sequential processing; however, the actual key is updated solely by the operating system to permit access of subsequent records. Therefore, the programmer does not write the ACTUAL KEY clause when sequential access is used.

The imperative statement in the AT END phrase associated with the next READ statement in order of execution is executed when the logical end of the mass storage file is detected. For WRITE statements, the detection of the logical end of a mass storage file before the execution of the CLOSE statement causes the actual key to contain an address outside the logical limits of the file. As this value represents an erroneous location in the file, the INVALID KEY path associated with a particular WRITE statement is executed when that verb is executed.

2.1.2. Random Access/Sequential Processing

With this technique, records are accessed in the order specified by the programmer in the ACTUAL KEY clause and processed sequentially (in the order in which they were accessed). The function of locating the data record in the file for subsequent reading or writing is accomplished by the SEEK statement. The SEEK statement is performed implicitly by a READ or WRITE if no immediately preceding SEEK has been executed, or if the SEEK and READ or WRITE refer to different records. The contents of the ACTUAL KEY are used by the compiler as the desired record's location identifier at the time of execution of the explicit of implicit SEEK statement.

Other procedural statements may be executed during the physical seeking operation if they have been written between the SEEK statement and the READ or WRITE statement for a particular file. The READ or WRITE of a particular record of a file cannot be executed until an explicit or implicit seek operation has been completed.

Until a READ statement is executed, any references to data items within the record description of the record being sought will refer to the contents of the last record obtained from the file. Therefore, if the program is written to take advantage of the ability to execute statements during the seek operation, this ''internal lag'' of one record must be taken into account by the programmer.

When random access is specified for a mass storage file, there is no logical end to the file. Thus, the INVALID KEY phrase must be specified for both the READ and WRITE statements. If, during execution of either a READ or a WRITE statement, the ACTUAL KEY points to a location outside the logical limits for a file, the imperative statement in the INVALID KEY phrase is executed.

2.2. FILE ORGANIZATION

The manner in which files are organized is usually a function of the individual mass storage system; the programmer need only select the access mode. However, since a general knowledge of file organization may prove helpful to the programmer, the following discussion is given.

Various types of file organization are possible for direct access devices. Since file organization can vary considerably with the individual implementation, only a general description of a few types can be given here.

Before discussing individual methods, it is important to distinguish between a physical record and a logical record.

A COBOL logical record is a group of related information, uniquely identifiable, and treated as a unit. In a COBOL program, an input or output statement refers to one logical record. A logical record may be contained within a single physical unit, or several logical records may be contained within a single physical unit, or a logical record may require more than one physical unit to contain it. The ACTUAL KEY refers to the physical location of a logical record in mass storage unit.

A physical record is a physical unit (or block) of information; its size and recording mode are convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware-dependent and bears no direct relationship to the size of the file of information contained on a device.

2.2.1. Sequential Data Organization

When sequential data organization is used, the logical records of a file are written sequentially (physically contiguous) in the order in which they are created. Readback is also sequential, that is, before a particular logical record can be accessed, all its predecessors must be read.

This type of data organization is normally used for tape and card files, but can also be used for mass storage files.

2.2.2. Relative Data Organization

Relative data organization uses relative logical record addressing. When this addressing scheme is used, the position of the logical records in a file is determined relative to the first record of the file; the maximum record number is defined by the size of the file. This structure is shown in Figure 2-1. A unique key (relative record address) identifies a record, enabling the user to access records in any sequence.

Files with relative data organization must be assigned to direct access devices.

Note that entries in this type of file consist entirely of data; no record identification entries are required for system recognition, thereby giving the user access to the contents of the entire file.

Two levels of classification are required with this type of organization in order to access a particular record: file-name and record.

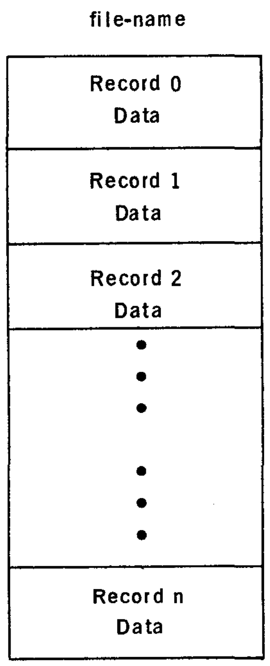

Figure 2-1. Relative File Structure

2.2.3. Direct Data Organization

Direct data organization is characterized by use of relative physical record (block) addressing. With this method, the location of each logical record in a file is determined by keys supplied by the programmer.

These keys specify two things:

- the block (relative to the first block of the file) at which the search is to begin
- the record sought

This is shown in Figure 2–2 where Block is the unique major data classification within a file and contains both data and block identification information. Record is the minor classification of data within a Block and contains both data and record identification information. Block and record identification information are for system use only and are not available to the programmer.

This type of organization permits the use of both fixed- and variable-length records and/or blocks. However, it requires three levels of classification: file-name, block, and record.

Files with direct data organization must be assigned to direct access devices.

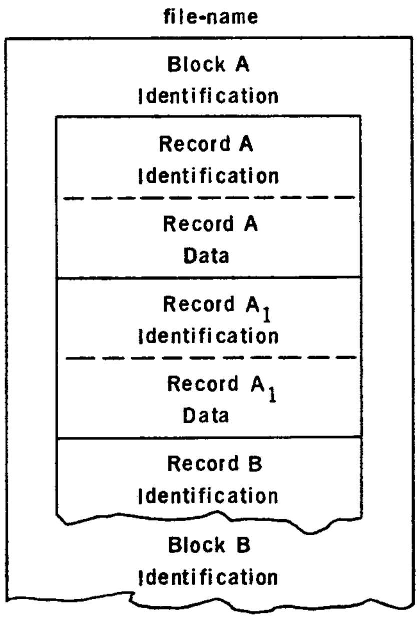

Figure 2-2. Direct File Structure

3. PROGRAMMING CONSIDERATIONS

3.1. GENERAL

The format and usage of the various entries required in the Environment, Data, and Procedure Divisions for utilization of the Mass Storage feature of COBOL are described in this section.

3.2. ENVIRONMENT DIVISION

There is no change required to the Configuration Section of this division when random access is used. However, the format of both the FILE-CONTROL and I-O-CONTROL paragraphs in the Input-Output Section differ from that given for sequential access in the *Fundamentals of COBOL – Language*.

3.2.1. FILE-CONTROL

Format:

```
              ⎧
FILE-CONTROL. ⎨ SELECT file-name
              ⎩
 ASSIGN TO [integer-1] implementor-name-1 [,implementor-name-2] . . .

 ⎡ , ⎧FILE-LIMIT IS  ⎫ ⎧data-name-1⎫            ⎧data-name-2⎫
 ⎣   ⎩FILE-LIMITS ARE⎭ ⎩literal-1  ⎭   THRU     ⎩literal-2  ⎭

 ⎡ , ⎧data-name-3⎫   THRU   ⎧data-name-4⎫ ⎤     ⎤
 ⎣   ⎩literal-3  ⎭          ⎩literal-4  ⎭ ⎦ . . .⎦

 , ACCESS MODE IS ⎧RANDOM    ⎫
                  ⎩SEQUENTIAL⎭

 , PROCESSING MODE IS SEQUENTIAL
                               ⎫
 , ACTUAL KEY IS data-name-5.  ⎬ . . .
                               ⎭
```

Description:

This paragraph names each file, identifies the file medium, and assigns each file to a particular hardware device. The FILE-CONTROL paragraph is required when the Input-Output Section header is present.

3.2.1.1. SELECT

Format:

$\underline{\text{SELECT}}\ \textit{file-name}$

Description:

Each file described in the Data Division must be named once and only once as the file-name in a separate SELECT clause in the FILE-CONTROL paragraph.

3.2.1.2. ASSIGN

Format:

$\underline{\text{ASSIGN}}\ \text{TO}\left[\textit{integer-1}\right]\ \textit{implementor-name-1}\ \left[,\ \textit{implementor-name-2}\right]\ldots$

Description:

All files used in a program must be assigned to an input or output medium. Integer-1 indicates the number of input-output units of a given medium assigned to the file-name specified in the SELECT clause. If integer-1 is not specified, the compiler determines the number of units assigned. The implementor-name of each input-output unit is given in the individual computer system's programmer reference manual. Integer-1 must be unsigned.

This clause is also used in some implementations in lieu of the FILE-LIMITS clause to implicitly define the limits of files. This is usually accomplished by either assigning one file to one device, or by using a control card to assign one or more files to a device. If file limit information is given in both the ASSIGN and FILE-LIMITS clause, either implicitly or explicitly, the value of the data items specified in the FILE-LIMITS clause must be within the range of limits specified in the ASSIGN clause.

3.2.1.3. FILE-LIMIT

Format:

$$\left[\ ,\left\{\begin{matrix}\underline{\text{FILE-LIMIT}}\ \underline{\text{IS}}\\ \underline{\text{FILE-LIMITS}}\ \underline{\text{ARE}}\end{matrix}\right\}\left\{\begin{matrix}\textit{data-name-1}\\ \textit{literal-1}\end{matrix}\right\}\ \underline{\text{THRU}}\ \left\{\begin{matrix}\textit{data-name-2}\\ \textit{literal-2}\end{matrix}\right\}\right.$$
$$\left.\left[\ ,\left\{\begin{matrix}\textit{data-name-3}\\ \textit{literal-3}\end{matrix}\right\}\ \underline{\text{THRU}}\ \left\{\begin{matrix}\textit{data-name-4}\\ \textit{literal-4}\end{matrix}\right\}\right]\ldots\right]$$

Description:

The FILE-LIMIT clause specifies the file limits within which logical records are to be obtained or placed. The two operands associated with the key word THRU represent the logical beginning and end of a mass storage file segment.

If the contents of the ACTUAL KEY data items point to records outside the given limits, the INVALID KEY phrase on READ and WRITE statements is executed.

3.2.1.4. ACCESS MODE

Format:

$, \underline{\text{ACCESS}}\ \text{MODE}\ \underline{\text{IS}}\ \left\{\begin{array}{l}\underline{\text{RANDOM}}\\ \underline{\text{SEQUENTIAL}}\end{array}\right\}$

Description:

The ACCESS MODE clause must be given for mass storage files. The key word RANDOM specifies that randomly located records are to be retrieved or written in accordance with the contents of the ACTUAL KEY clause. The key word SEQUENTIAL specifies that records are to be obtained or placed sequentially; that is, the next logical record is made available from or placed in a file upon execution of a READ or WRITE statement, respectively.

3.2.1.5. PROCESSING MODE

Format:

$, \underline{\text{PROCESSING}}\ \text{MODE}\ \underline{\text{IS}}\ \underline{\text{SEQUENTIAL}}$

Description:

This clause is required for mass storage files; it specifies that records are to be processed in the order in which they were accessed.

3.2.1.5. ACTUAL KEY

Format:

$, \underline{\text{ACTUAL}}\ \text{KEY}\ \underline{\text{IS}}\ \textit{data-name-5}$

Description:

This phrase is required when ACCESS MODE IS RANDOM, since the contents of data-name-5 are used by the SEEK, READ, and WRITE statements to locate a specific mass storage record. Therefore, the address or a pointer to the address of the record must be placed in data-name-5 prior to the execution of a SEEK statement (or the implicit SEEK statement contained in READ and WRITE statements). The ACTUAL KEY clause is not used when ACCESS MODE IS SEQUENTIAL.

Upon execution of a READ, the logical record specified by data-name-5 is made available from the file. When executing a WRITE, the specified logical record is placed in the file location specified by data-name-5.

3.2.2. I-O-CONTROL

Format:

```
                ┌                ┌       ┌ file-name-1        ┐┐
I-O-CONTROL.    │ ; RERUN        │ ON    └ implementor-name   ┘┘
                └

                ┌ integer-1 RECORDS OF file-name-2 ┐ ┐
        EVERY   │ integer-2 CLOCK-UNITS            │ │ . . .
                └ condition-name                   ┘ ┘

      [; SAME [RECORD] AREA FOR file-name-3

       {, file-name-4} . . . ] . . . .
```

Description:

The I-O-CONTROL paragraph is optional; it is used to specify input-output techniques, rerun points, and memory area to be shared by different files.

3.2.2.1. RERUN

Format:

```
┌          ┌     ┌ file-name-1       ┐┐
│ ; RERUN  │ ON  └ implementor-name  ┘┘
└
            ┌ integer-1 RECORDS OF file-name-2 ┐ ┐
    EVERY   │ integer-2 CLOCK-UNITS            │ │ . . .
            └ condition-name                   ┘ ┘
```

Description:

The RERUN clause specifies where and when rerun information is to be recorded. This information (called a checkpoint record) is a memory dump taken at a given point in the computer run during execution of the object program. The checkpoint record contains all information required to rerun the program from that given point. This permits processing to be resumed from the last checkpoint in the program (rather than from the beginning of the run) in the event of program stoppage due to an error or interruption.

Memory can be dumped into an output file by specifying file-name-1, or into a separate rerun file by specifying the implementor-name of the device.

If file-name-1 is specified in the ON option, normal closing functions are performed for this file along with the memory dump. In this case, file-name-2 can be either an input or an output file.

The EVERY portion of the RERUN clause controls the intervals between memory dumps as follows:

- When the number of records specified by integer-1 of an input or output file, file-name-2, have been processed. In this case, implementor-name must be specified.

- When the interval of time specified by integer-2 in the CLOCK-UNITS option has elapsed. The unit of time is dependent on the implementation for the particular computer. Here also, implementor-name must be specified.

- When a hardware switch assumes the status specified by condition-name. In this case, the hardware switch must be defined in the SPECIAL-NAME paragraph of the Configuration Section in the Environment Division. The status of the switch is interrogated at the intervals specified in the implementation for the particular computer.

3.2.2.2. SAME

Format:

```
[; SAME [RECORD] AREA FOR file-name-3
     {, file-name-4} . . .] . . .  .
```

Description:

The SAME AREA clause specifies that two or more files are to use the same memory area during processing. The area being shared includes all storage areas (including alternate areas) assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time.

The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of:

- each opened output file with its file-name appearing in this SAME RECORD AREA clause,
- the most recently read input file with its file-name appearing in this SAME RECORD AREA clause.

More than one SAME clause may be included in a program. However:

- A file-name must not appear in more than one SAME AREA clause.

- A file-name must not appear in more than one SAME RECORD AREA clause.

- If one or more file-names appearing in a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, additional file-names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause. The rule that only one of the files mentioned in a SAME AREA clause can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA clause can be open at any given time.

A file-name that represents a sort file must not appear in the SAME clause unless the RECORD option is used; see *Fundamentals of COBOL–Sorting.*

3.3. DATA DIVISION

The format and usage of the File-Section of the Data Division are the same for both mass storage and sequential files. Therefore, only the format and a brief description are given here; see *Fundamentals of COBOL – Language* for a complete description of the File-Section.

3.3.1. File Description

Format:

```
FD file-name
   [; BLOCK contains [integer-1 TO] integer-2 {RECORDS   }]
                                              {CHARACTERS}]

   [; DATA {RECORD IS  } data-name-1 [, data-name-2] . . . ]
           {RECORDS ARE}

    ; LABEL {RECORD IS  } {STANDARD                           }
            {RECORDS ARE} {OMITTED                            }
                          {data-name-3 [, data-name-4] . . .  }

   [; RECORD ; BLOCK CONTAINS [integer-3 TO] integer-4 CHARACTERS]

   [; VALUE OF data-name-5 IS {data-name-6}
                              {literal-1  }

   [, data-name-7 IS {data-name-8} ] . . . ] .
                     {literal-2  }
```

Description:

The File Description paragraph identifies a given file, the records contained therein, and describes the physical structure of that file.

3.4. PROCEDURE DIVISION

The Declaratives portion of the Procedure Division and the verb formats used with the Mass Storage feature are described in the following paragraphs. Note that the Declaratives portion is the same for the Sequential Access and Random Access modules of USA Standard COBOL.

3.4.1. DECLARATIVES

Format:

```
DECLARATIVES.

{section-name SECTION.  declarative-sentence
{paragraph-name. { sentence } . . . } . . . } . . .

END DECLARATIVES.
```

Description:

DECLARATIVES is a set of one or more sections written at the beginning of the Procedure Division. Each section contains a compiler-directing statement which specifies the circumstances under which the procedures contained therein are to be executed. DECLARATIVES exists outside the main body of the Procedure Division at execution time and is used only when the condition defined in the USE statement contained in the declarative-sentence arises.

Each declarative operates under control of either the inline procedure or the input/output system. The Declaratives portion of the Procedure Division must be preceded by the header DECLARATIVES and terminated by the key words END DECLARATIVES.

Each declarative must begin with a section-name followed by a declarative-sentence (USE statement). The remainder of the section consists of one or more procedural paragraphs.

3.4.1.1. USE

Formats:

Format 1:

```
USE AFTER STANDARD ERROR PROCEDURE ON

  ( file-name-1 [, file-name-2] . . . )
  ( INPUT                             )
  ( OUTPUT                            )
  ( I-O                               )
```

Format 2:

```
     {BEFORE}            [BEGINNING]
USE  {AFTER }  STANDARD  [ENDING   ]  [FILE] LABEL PROCEDURE ON

{ file-name-1 [, file-name-2] . . . }
{ INPUT                             }
{ OUTPUT                            }
{ I-O                               }
```

Description:

The USE statement is used to specify special procedures for input and output label and error handling.

The USE statement, when present, must immediately follow a declarative section header and be followed by a period followed by a space. The remainder of the declarative must consist of one or more procedural paragraphs that define the procedure to be used.

The USE statement is not an executable statement; rather, it defines conditions calling for the execution of its associated procedures.

The designated procedures are executed by the input/output system at the appropriate time as follows:

- Format 1 is executed after completion of the standard input/output error routine.
- Format 2 is executed before or after a beginning or ending input label check procedure is executed; before a beginning or ending output label is created; after a beginning or ending output label is created, but before it is written on the file; before or after a beginning or ending input/output label check procedure is executed. (Note that Format 2 USE procedures never apply to files that are described with the LABEL RECORDS ARE OMITTED clause.)

A file-name may appear in more than one USE statement providing it does not cause simultaneous requests for execution of more than one USE declarative.

No references are permitted within a USE statement to any non-declarative procedure; conversely, no nondeclarative procedure can make reference to a procedure-name which appears in the Declaratives portion, with the following exception: a PERFORM statement may refer to a USE declarative or its associated procedures.

The following rules apply to Format 2:

- If an option other than file-name-1 is specified, references within procedures to common label items need not be qualified by a file-name. A common label item is an elementary data item that appears in every label record of the program, but does not appear in any of the program's data records. A common label item must have the same name, description, and relative position in every label record.
- If the file-name option is selected, the File Description entry for file-name-1 must not contain a LABEL RECORDS ARE OMITTED clause. If the INPUT, OUTPUT, or I-O option is specified, the USE procedures do not apply to any of their respective file types which are described with the LABEL RECORDS ARE OMITTED clause.
- If the key words BEGINNING or ENDING are omitted, the designated procedures are executed for both beginning and ending labels.
- The designated procedures for FILE labels are executed even if the word FILE is not specified.

3.4.2. Verbs

The COBOL Mass Storage feature requires the use of modified formats for the CLOSE, OPEN, READ, and WRITE verbs and the addition of a new verb, SEEK.

3.4.2.1. CLOSE

Format:

```
CLOSE file-name-1 [UNIT][WITH LOCK]
[, file-name-2 [UNIT][WITH LOCK]] . . .
```

Description:

The CLOSE statement terminates processing of the file(s) specified by file-name. Once a file has been closed, no other statement can be executed for that file until it is again opened.

The WITH LOCK option locks the file (or unit), thereby preventing it from being opened again during execution of the current object program. The UNIT option is only applicable to mass storage files in the sequential access mode (see *Fundamentals of COBOL – Language* for closing of sequential files).

3.4.2.2. OPEN

Format:

```
       ⎧ INPUT file-name [, file-name] . . .   ⎫
OPEN   ⎨ OUTPUT file-name [, file-name] . . .  ⎬  . . .
       ⎩ I-O file-name [, file-name] . . .     ⎭
```

Description:

The OPEN statement initiates processing of the named files by checking and/or writing labels, and performing any other input/output operations prior to accessing the first record in a given file. The address of the initially accessed record is supplied through the FILE-LIMIT clause in the Environment Division. However, the OPEN statement does not obtain or release the first data record; a READ or a WRITE statement must be executed. The contents of the data-names specified in the FILE-LIMIT clause of the FILE-CONTROL paragraph are checked only when the OPEN statement is executed.

Each of the choices (INPUT, OUTPUT, I-O) can be specified only once in an OPEN statement. A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.

When checking or writing the first label, the user's beginning label subroutine is executed if specified in a USE statement.

If an input file (ACCESS MODE IS SEQUENTIAL) is designated as OPTIONAL in the FILE-CONTROL paragraph, the object program causes an interrogation for the presence of this file. If the file is not present, the first READ statement for this file causes the imperative statement in the AT END phrase to be executed. See the *Fundamentals of COBOL – Language* for a description of the OPTIONAL and AT END clauses.

The I-O option permits the opening of a mass storage file for both input and output operations. This option cannot be used if the mass storage file is being initially created, since its presence implies previous existence of the file.

The I-O option causes the following to occur:

- The label is checked in accordance with the implementor's specified conventions for input/output label checking.
- The user's beginning label subroutine is executed if one is specified by a USE statement.
- The new label is written according to the implementor-specified conventions.

3.4.2.3. READ

Formats:

Format 1:

READ *file-name* RECORD [INTO *identifier*] ; AT END *imperative-statement*

Format 2:

READ *file-name* RECORD [INTO *identifier*] ; INVALID KEY *imperative-statement*

Description:

Format 1 of the READ statement is used only for non-mass storage files or for mass storage files in the sequential access mode. It makes available the next logical record from an input file (file-name) and allows performance of imperative-statement when end of file is detected. The rules for this format are given in *Fundamentals of COBOL – Language*.

Format 2 is used to read mass storage files in the random access mode. It makes the record specified in the ACTUAL KEY clause available and allows performance of imperative-statement if the contents of the ACTUAL KEY data item are found to be invalid. The rules for this format are given in the following paragraphs.

An OPEN statement must be executed for a file prior to the execution of the first READ statement for that file. The record accessed by a READ statement is the record available in the input area prior to execution of any statement following that READ statement.

If a file contains more than one type of logical record (that is, more than one 01 level record description entry), all records in the file share the same record area, with the area being implicitly redefined for each record. Only the information present in the current record is accessible.

The INTO option can only be used if the input file contains records of one type. The storage areas associated with file-name and identifier, respectively, must be separate areas. File-name must not represent a sort file.

If the INTO option is used, the current record is moved from the input area to the area specified by identifier in accordance with the rules for the MOVE statement without the CORRESPONDING option. This record is now available in the identifier data area and the input record area.

The READ statement performs the functions of the SEEK statement implicitly unless a SEEK statement referencing the same record has been executed prior to the READ statement.

3.4.2.4. SEEK

Format:

<u>SEEK</u> *file-name* RECORD

Description:

The SEEK statement initiates the access of a mass storage data record for subsequent reading or writing. The data-name in the ACTUAL KEY clause contains the location of the record sought. At execution time, the contents of the ACTUAL KEY are tested for validity. If invalid, the imperative-statement in the INVALID KEY phrase of the next executed READ or WRITE statement for the associated file is executed.

Two SEEK statements for the same mass storage file may logically be executed without an intervening READ or WRITE statement. In this case, the validity check associated with the first SEEK statement is negated.

The implied SEEK in a READ or WRITE statement is not performed if the READ or WRITE is preceded by a separate SEEK statement referencing the same record as the READ or WRITE.

3.4.2.5. WRITE

Format:

<u>WRITE</u> *record-name* [<u>FROM</u> *identifier*] ; <u>INVALID</u> KEY *imperative-statement*

Description:

This format is used to write mass storage files at the location specified in the ACTUAL KEY clause in either the random or sequential access mode. It releases a logical record to an output file and permits performance of an imperative-statement if the file-limit is exceeded (ACTUAL KEY data item is found to be invalid). The rules for this format are given in the following paragraphs.

An OPEN statement must be executed for a file prior to the execution of the first WRITE statement for that file.

The logical record release by the execution of the WRITE statement is no longer available unless the associated file is named in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated output file.

If the FROM option is specified, the data is moved from the area specified by identifier to the output area according to the rules specified for the MOVE statement without the CORRESPONDING option. After execution of the WRITE statement, the information in identifier is available, even though that in record-name is not.

The storage areas for record-name and identifier, respectively, must be separate areas. Record-name must not represent a sort file. Record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

The WRITE statement implicitly performs the function of the SEEK statement for a specific mass storage record, unless a SEEK statement was executed for this record prior to the execution of the WRITE statement. The imperative-statement in the INVALID KEY phrase is executed when the contents of the actual key being used to obtain the mass storage record are found to be invalid. When the INVALID KEY condition exists, no writing takes place and the information in the record area is available. End of file conditions are detected by means of the INVALID KEY phrase.

If a mass storage file is contained on more than one mass storage unit, end of unit procedures are the responsibility of the programmer, who must take the appropriate action consistent with system procedures to effect the transfer of the write operation from one unit to the next.

Term	Reference	Page
A		
ACCESS MODE,		
ACTUAL KEY	3.2.1.5	3–3
description of	3.2.1.4	3–3
format of	3.2.1.4	3–3
in FILE-CONTROL	3.2.1	3–1
ACTUAL KEY,		
ACCESS MODE	3.2.1.4	3–3
FILE-LIMIT	3.2.1.3	3–2
format of	3.2.1.5	3–3
in FILE-CONTROL	3.2.1	3–1
in random access/sequential processing	2.1.2	2–2
in sequential access/ sequential processing	2.1.1	2–1
logical record designation	2.2	2–2
READ	3.4.2.3	3–11
SEEK	3.4.2.3	3–12
WRITE	3.4.2.5	3–12
AFTER	3.4.1.1	3–7, 3–8
ASSIGN,		
description of	3.2.1.2	3–2
format of	3.2.1.2	3–2
in FILE-CONTROL	3.2.1	3–1
Asynchronous Processing	See Random Processing	
AT END,		
in sequential access/ sequential processing	2.1.1	2–1
OPEN	3.4.2.2	3–10
READ	3.4.2.3	3–11
B		
BEFORE	3.4.1.1	3–8
BEGINNING	3.4.1.1	3–8
BLOCK,		
addressing of	2.2.3	2–4
definition of	2.2	2–2
BLOCK CONTAINS	3.3.1	3–6
C		
CHARACTERS,		
CLOCK CONTAINS	3.3.1	3–6
RECORD	3.3.1	3–6
Checkpoint Record	See Record	

Term	Reference	Page
CLOCK-UNITS	3.2.2.1	3–4
CLOSE,		
description of	3.4.2.1	3–9
format of	3.4.2.1	3–9
in sequential access/ sequential processing	2.1.1	2–1
OPEN	3.4.2.2	3–10
COBOL Mass Storage Feature,		
file handling	2.1	2–1
programming considerations	3.1	3–1
purpose of	1.1	1–1
verb formats	3.4	3–6
CONFIGURATION SECTION	3.2	3–1
D		
DATA DIVISION	3.3	3–6
DATA RECORD	3.3.1	3–6
DECLARATIVES,		
description of	3.4.1	3–7
format of	3.4.1	3–7
Direct Access,		
in file organization	2.2	2–2
storage devices	1.1	1–1
Direct Data Organization	2.2.3	2–4
Direct File Structure	See Direct Data Organization	
E		
END DECLARATIVES	3.4.1	3–7
ENDING	3.4.1.1	3–8
ENVIRONMENT DIVISION	3.2	3–1
ERROR PROCEDURE	3.4.1.1	3–7
EVERY	3.2.2.1	3–4
F		
FD	See File Description	
FILE	3.4.1.1	3–8
File Description	3.3.1	3–6
File Handling	2.1	2–1

Term	Reference	Page
File Organization	2.2	2–2
FILE CONTROL,		
description of	3.2.1	3–1
format of	3.2.1	3–1
FILE-LIMIT,		
ASSIGN	3.2.1.2	3–2
description of	3.2.1.3	3–2
format of	3.2.1.3	3–2
in FILE-CONTROL	3.2.1	3–1
OPEN	3.4.2.1	3–10
File-name,		
direct data organization	2.2.3	2–4
relative data organization	2.2.2	2–3
FILE SECTION,		
description of	3.3.1	3–6
format of	3.3.1	3–6
FROM	3.4.2.5	3–12
I		
INPUT,		
OPEN	3.4.2.2	3–10
USE	3.4.1.1	3–7, 3–8
INPUT-OUTPUT SECTION	3.2	3–1
INTO	3.4.2.3	3–11
INVALID KEY,		
FILE-LIMIT	3.2.1.3	3–2
in random access/sequential processing	2.1.2	2–2
in sequential access/ sequential processing	2.1.1	2–1
SEEK	3.4.2.4	3–12
WRITE	3.4.2.5	3–12
I-O,		
OPEN	3.4.2.2	3–10
USE	3.4.1.1	3–7, 3–8
I-O-CONTROL,		
description of	3.2.2	3–4
format of	3.2.2	3–4
L		
LABEL PROCEDURE	3.4.1.1	3–8
LABEL RECORD	3.3.1	3–6

Term	Reference	Page
M		
Mass Storage	1.1	1–1
Mass Storage File,		
updating of	2.1.1	2–1
usage of	2.1.1	2–1
Memory Dump	3.2.2.1	3–4
MOVE,		
READ	3.4.2.3	3–11
WRITE	3.4.2.5	3–12
O		
OMITTED	3.3.1	3–6
ON,		
RERUN	3.2.2.1	3–4
USE	3.4.1.1	3–8
OPEN,		
description of	3.4.2.2	3–10
format of	3.4.2.2	3–10
READ	3.4.2.3	3–11
WRITE	3.4.2.5	3–12
OPTIONAL	3.4.2.2	3–10
OUTPUT,		
OPEN	3.4.2.2	3–10
USE	3.4.1.1	3–7, 3–8
P		
PERFORM	3.4.1.1	3–8
Physical Unit	See Record	
PROCEDURE DIVISION	3.4	3–6
PROCESSING MODE,		
description of	3.2.1.5	3–3
format of	3.2.1.5	3–3
in FILE-CONTROL	3.2.1	3–1
Programming Considerations	3.1	3–1
R		
RANDOM	3.2.1.4	3–3
Random Access,		
definition of	1.2	1–1
storage devices	1.1	1–1
with sequential processing	2.1.2	2–2
Random Processing	1.2	1–1

Term	Reference	Page
READ,		
ACCESS MODE	3.2.1.4	3-3
ACTUAL KEY	3.2.1.5	3-3
description of	3.4.2.3	3-11
FILE-LIMIT	3.2.1.3	3-2
format of	3.4.2.3	3-11
in random access/sequential processing	2.1.2	2-2
in sequential access/ sequential processing	2.1.1	2-1
OPEN	3.4.2.2	3-10
SEEK	3.4.2.4	3-12
Record,		
addressing	2.2.3	2-4
checkpoint	3.2.2.1	3-4
logical	2.2	2-2
physical	2.2	2-2
RECORD,		
BLOCK CONTAINS	3.3.1	3-6
READ	3.4.2.3	3-11
RERUN	3.2.2.1	3-4
SEEK	3.4.2.4	3-12
RECORD CONTAINS	3.3.1	3-6
Relative Data Organization,		
definition of	2.2.2	2-3
file structure	2.2.2	2-3
Relative File Structure	See Relative Data Organization	
RERUN,		
description of	3.2.2.1	3-4
format of	3.2.2.1	3-4
in I-O-CONTROL	3.2.2	3-4
S		
SAME,		
description of	3.2.2.2	3-5
format of	3.2.2.2	3-5
in I-O-CONTROL	3.2.2	3-4
WRITE	3.4.2.5	3-12
SAME AREA	See SAME	
SAME RECORD AREA	See SAME	
SECTION	3.4.1	3-7
SEEK,		
ACTUAL KEY	3.2.1.5	3-3
description of	3.4.2.4	3-12
format of	3.4.2.4	3-12
implicit	3.2.1.5	3-3
in random access/sequential processing	2.1.2	2-2
READ	3.4.2.3	3-11
WRITE	3.4.2.5	3-13
SELECT,		
description of	3.2.1.1	3-2
format of	3.2.1.1	3-2
in FILE-CONTROL	3.2.1	3-1

Term	Reference	Page
SEQUENTIAL,		
ACCESS MODE	3.2.1.4	3-3
PROCESSING MODE	3.2.1.5	3-3
Sequential Access,		
definition of	1.2	1-1
with sequential processing	2.1.1	2-1
Sequential Data Organization	2.2.1	2-3
Sequential Processing,		
definition of	1.2	1-1
with random access	2.1.2	2-2
with sequential access	2.1.1	2-1
SPECIAL-NAMES	3.2.2.1	3-4
STANDARD,		
LABEL RECORD	3.3.1	3-6
USE	3.4.1.1	3-7
Synchronous Processing	See Sequential Processing	
Terminology	1.2	1-1
T		
THRU	3.2.1.3	3-2
TO	3.3.1	3-6
U		
UNIT	3.4.2.1	3-9
USE,		
DECLARATIVES	3.4.1	3-7
description of	3.4.1.1	3-8
format of	3.4.1.1	3-7
OPEN	3.4.2.2	3-10
V		
VALUE OF	3.3.1	3-6
Verbs	3.4.2	3-9
W		
WITH LOCK	3.4.2.1	3-9
WRITE,		
ACCESS MODE	3.2.1.4	3-3
ACTUAL KEY	3.2.1.5	3-3
description of	3.4.2.5	3-12
FILE-LIMIT	3.2.1.3	3-2
format of	3.4.2.5	3-12
in random access/sequential processing	2.1.2	2-2
in sequential access/ sequential processing	2.1.1	2-1
OPEN	3.4.2.2	3-10
SEEK	3.4.2.4	3-12