OS 1100

UNISYS

ASCII COBOL

Supplementary Reference Manual

Relative to Release 5R1 January 1988

Printed in U S America
Priced Item UP-8584 Rev. 1

Contents

Section 1
Introduction

This supplementary reference manual documents items that are currently in use in ASCIl COBOL but lie outside the mainstream of COBOL language features. They remain a part of the ASCII COBOL system to assist users in converting to American National Standard COBOL X3.23-1974. Their use in new applications is discouraged because future OS 1100 COBOL systems will not provide these items.

Note: *Existing programs. when updated should be modified to exclude the use of the items documented in this manual.*

This document contains only the nonstandard items of ASCll COBOL. For more information regarding ASCll COBOL, see the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

References to the American National Standard COBOL of X3.23-1968 and X3.23-1974 in the remainder of this manual are shortened to Standard COBOL 1968 and Standard COBOL 1974, respectively.

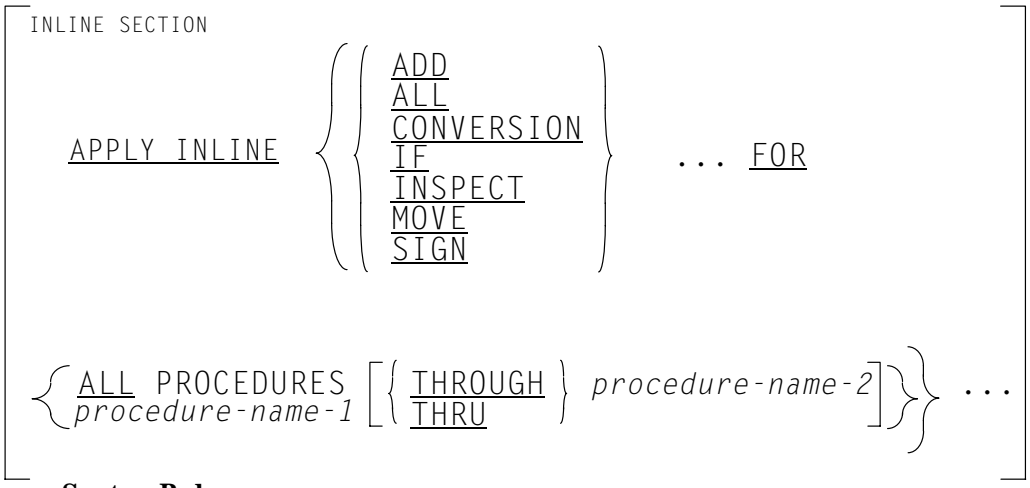
Section 2
Nucleus

2.1. Inline Section in the Control Division

Function

The Inline Section is obsolete since the compiler automatically generates inline code.

Format



Syntax Rules

1. The words THRU and THROUGH are equivalent.
2. Within the source program, *procedure-name-1* must appear before *procedure-name-2*.

General Rules:

1. The Inline Section is obsolete since inline code is automatically generated by the compiler for all the cases mentioned in the following general rules.

2. The functions for which inline code can be generated within specific procedures of the Procedure Division ore INSPECT, MOVE, IF, ADD, SIGN, and CONVERSION. These functions do not apply to array nonsynchronized (all occurrences do not have the same starting byte) data items. If the functions use a mixture of Fieldata (FD) and ASCII data items. special code will not be generated. The CONVERSION, INSPECT, and SIGN functions will generate special code for ASCII usage data items only. If one of the operands in the function is of variable length, inline code will not be generated.
3. The procedures for which the inline functions will be generated depend on the *procedure-names* specified in the FOR clause. It should be noted that the functions apply to the physical location of the procedures, not the logical locations.
4. If ALL PROCEDURES is specified. the inline functions specified will be generated for the entire Procedure Division.
5. If only *procedure-name-1* is specified. the inline functions apply only to that procedure.
6. If *procedure-name-1* THROUGH *procedure-name-2* is specified, the inline functions apply to all procedures starting at *procedure-name-1* and continuing through *procedure-name-2*.
7. If *procedure-name-1* is specified but not defined within the Procedure Division, no inline code will be generated.
8. If *procedure-name-2* is specified but not defined within the Procedure Division, inline code will be generated starting at *procedure-name-1* and ending at the last line of the source program.
9. If *procedure-name-2* appears before *procedure-name-1*, inline code will be generated as if *procedure-name-2* were not defined.
10. When the APPLY INLINE ADD clause is specified, inline code will be generated for all additions and subtractions where both operands are unsigned and less than two words in length.
11. When the APPLY INLINE ALL clause is specified, it includes all the special functions described in this section.
12. When the APPLY INLINE CONVERSION clause is specified, inline code will be generated for the following conversions:
    a. Converting from a binary (USAGE IS COMPUTATIONAL) item to a numeric character (PIC 9 DISPLAY) item where they are either signed or unsigned and are less than five characters in length.
    b. Converting from a numeric character item to a binary item where they are either signed or unsigned and less than six characters in length.
13. When the APPLY INLINE IF clause is specified. inline code will be generated for all comparisons where both fields have equal start bytes, are less than 512 bytes in length, and are equal in length, or if after scaling and padding, two items of different lengths are less than two words in length.

14. When the APPLY INLINE INSPECT clause is specified, INSPECT statements with any of the following attributes will not generate inline code:
    a. The item being examined is described as a signed numeric item.
    b. Any tallying or replacing items ore not single character items.
    c. Multiple tallying items ore stated.
    d. Multiple replacing clauses are stated.
    e. The UNTIL FIRST option is used.
    f. A BEFORE/AFTER INITIAL clause is specified.
15. When the APPLY INLINE MOVE clause is specified, inline code will be generated for alphanumeric moves when the sending and receiving fields have equal start bytes and the size of the receiver is less than 512 bytes in length. If both sender and receiver start at byte 0, there is no restriction on length.
16. When the APPLY INLINE SIGN clause is specified, inline code will be generated for trailing overpunch sign extractions and stored for signed numeric items.

2.2. The REMARKS Paragraph in the Identification Division

Format

```
[REMARKS. [comment-entry] ...]
```

Description

This paragraph contains a general description of the program. A *comment-entry* may be made up of any combination of characters from the allowable COBOL character set.

Standard COBOL 1974 has replaced the REMARKS paragraph by a generalized comment facility. An asterisk (*) in character position 7 identifies any line as a comment line. A slash (/) in character position 7 causes the line to be treated as a comment and causes page ejection. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

2.3. Data Division

2.3.1. Data Division Structure

The fixed names of the sections in their required order of appearance as section headers in the Data Division are:

FILE SECTION
COMMUNICATION SECTION
WORKING-STORAGE SECTION
COMMON-STORAGE SECTION
LINKAGE SECTION
REPORT SECTION
FILE SECTION
COMMON-STORAGE SECTION
WORKING-STORAGE SECTION
LINKAGE SECTION
COMMUNICATION SECTION
REPORT SECTION

2.3.2. Reserved Data-Names

Special registers are data-names with implicit data descriptions. Since special registers are merely special predefined data-names, they may be used in any context where a data-name is legal. However. since these special registers have particular functions during execution, modification of their values should be done only with full realization of the impact on subsequent processing. It is illegal to explicitly define any special register data-name.

2.3.2.1. MCFLAG Register

The word MCFLAG is the name of a special register whose implicit definition is that of a 36-bit binary integer, initially set equal to 1. MCFLAG is defined for all COBOL programs but is intended to allow dynamic suppression of output from the MONITOR statement. (See 2.4.6.)

Setting MCFLAG to zero will suppress monitor output; thus, by setting and resetting MCFLAG to zero and nonzero values, the user can dynamically inhibit monitor printing.

2.3.2.2. TALLY Register

The word TALLY is the name of a special register whose implicit description is that of a 36-bit binary integerù TALLY is defined for all COBOL programs. The primary use of the TALLY register is to hold information produced by the EXAMINE Statement. (See 2.4.3.) The word TALLY may also be used as a data-name wherever an elementary data item of integral value may appear.

2.3.2.3. PICTURE “H”

An “H” represents COMPUTATIONAL usage. Only one “H” is allowed in a PICTURE. It must appear as the leftmost character except in a signed item, in which case “H” must immediately follow “S”. The “H” is not used in determining the size of an item.

2.3.3. The POINT LOCATION Clause

Function

The POINT LOCATION clause states the decimal scaling for an exact binary item.

General Format

```
[ ; POINT LOCATION is { LEFT  } integer-1 { PLACE  } ]
                      { RIGHT }           { PLACES }
```

Syntax Rules

1. The POINT LOCATION clause can only be used with an exact binary PICTURE. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).
2. In the absence of the POINT LOCATION clause, an exact binary data item is assumed to be an integer.
3. The parameter *integer-1* is the decimal scale of the data-name.

2.3.4. The USAGE Clause

Function

The USAGE clause specifies the format of a Fieldata item in storage. For all other valid usages and rules, see the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

General Format

```
[USAGE IS] { COMPUTATIONAL-4 }
           { COMP-4          }
           { DISPLAY-1       }
           { DISP-1          }
```

Syntax Rules

1. The PICTURE character-string of a COMPUTATIONAL-4 item can contain only 9’s, the operational sign character “S”, the implied decimal point character “V”, or one or more P’s (see the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version)).
2. COMP-4 and DISP-1 are abbreviations for COMPUTATIONAL-4 and DISPLAY-1, respectively.

Description

If the group usage is defined as DISPLAY-1 or COMPUTATIONAL-4, the group is considered to be an alphanumeric string of Fieldata characters. If the group usage is not explicitly defined, the group is considered to be an alphanumeric string of ASCII characters unless the first contained elementary non-FILLER item has USAGE DISPLAY-1 or COMPUTATIONAL-4, in which case the group is considered to be an alphanumeric string of Fieldata characters.

When mixing ASCII and Fieldata items in the same group. each group item must have explicit usage declared in order to ensure proper data allocation. There are two general cases where a group containing exact binary or 6-bit items must have an explicit 6-bit usage to ensure proper allocation:

- The group item does not start on a half-word boundary.
- An occurring group item which contains a 6-bit item does not both start and end on a half-word boundary.

A COMPUTATIONAL-4 item is capable of representing a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL-4, the elementary items in the group are COMPUTATIONAL-4. The group item itself is not COMPUTATIONAL-4 (cannot be used in computations).

DISPLAY-1 and COMPUTATIONAL-4 are aligned on sixth-word boundaries. The meaning and allocation of DISPLAY-1 and COMPUTATIONAL-4 are as follows:

Usage	Meaning	Allocation
DISPLAY-1	Fieldata characters	6 bits, 1/6 word, per allocation character represented in PICTURE clause.
COMPUTATIONAL-4	Fixed-point binary: Also called Fieldata sync-binary	1 digit, 1/6 word, 6 bits 2-3 digits, 1/3 word, 12 bits 4-5 digits, 1/2 word, 18 bits 6 digits, 2/3 word, 24 bits 7-8 digits, 5/6 word, 30 bits 9-10 digits, 1 word, 36 bits 11-12 digits, 1-1/6 words, 42 bits 13-14 digits, 1-1/3 words, 48 bits 15 digits, 1-1/2 words, 54 bits 16-17 digits, 1-2/3 words, 60 bits 18 digits, 1-5/6 words, 66 bits

When the usage of the sender is Fieldata and the receiver is ASCII or vice versa, character conversion is performed for MOVE statements. Also, if the move is such that Fieldata or ASCII numerics are moved to a computational data item and conversion and truncation are required for that move, the truncation will be binary. That is, the Fieldata to ASCII conversion, or vice versa, will be performed before the truncation.

When mixing USAGE clauses in the STRING statement, it should be noted that the processing time will be increased because of the required conversions between ASCII and Fieldata.

2.4. Procedure Division

2.4.1. The ACCEPT Statement

Function

The ACCEPT statement causes low-volume data to be made available to the specified data item.

Format 1

```
ACCEPT identifier [ FROM { CARD-READER } ]
                         { CONSOLE     }
```

Format 2

```
ACCEPT identifier FROM DATE-TIME
```

General Rules

Format 1

See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

Format 2

DATE-TIME is composed of the data elements month, day, year, hour, minutes, seconds, with each value being a 2-character number. Month will contain 01 through 12; day 01 through 31; year 00 through 99; hour 00 through 23; minute and second 00 through 59.

DATE-TIME, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item 12 digits in length.

2.4.2. The DISPLAY Statement

Function

The DISPLAY statement causes low-volume data to be transferred to an appropriate hardware device.

Format

Description

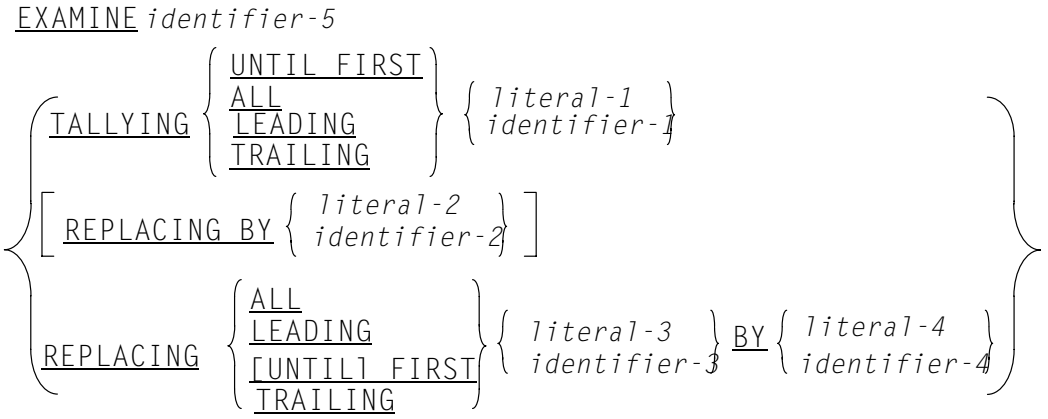
See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

2.4.3. The EXAMINE Statement

Function

The EXAMINE statement replaces or counts the number of occurrences of a given character in a data item.

Format



Description

The description of *identifier-5* must have USAGE DISPLAY or DISPLAY- 1, but may have any legal picture for USAGE DISPLAY or DISPLAY-1 items.

The items *identifier-1*, *identifier-2*. *identifier-3*, *identifier-4*, *literal-1*, *literal-1*, *literal-2*, *literal-3*. and *literal-4* must be single characters and belong to a class consistent with that of *identifier-5*. Literals may be any figurative constant except ALL.

Examination starts with the leftmost character in *identifier-5* and proceeds character-by-character to the right, unless TRAILING is specified, in which case examination begins with the rightmost character and proceeds to the left.

If *identifier-5* is an elementary numeric item, it must consist of numeric characters and may possess an operational sign (e.g., an overpunch on the low order character). If the letter "S" is used in the PICTURE character-string of the data item description to indicate the presence of an operational sign, the sign is completely ignored by the EXAMINE statement.

Nonnumeric items may contain any character in the COBOL character set.

The TALLYING phrase creates an integral count which replaces the value of a special register called TALLY. (See 2.3.2.2.) The count represents the number of:

1. Occurrences of *literal-1* or *identifier-1* when the ALL phrase is used.
2. Occurrences of *literal-1* or *identifier-1* prior to encountering a character other than *literal-1* or *identifier-1* when the LEADING phrase is used.
3. Characters not equal to *literal-1* or *identifier-1* encountered before the first occurrence of *literal-1* or *identifier-1* when the UNTIL FIRST phrase is used.
4. Occurrences of *literal-1* or *identifier-1* prior to encountering a character other than *literal-1* or *identifier-1* when the TRAILING phrase is used.

When TALLYING and REPLACING BY are both used, replacing occurs only within the examining scope of the TALLYING phrase.

The REPLACING phrase without TALLYING does not change the current contents of TALLY. The rules for either type of REPLACING are:

1. When the ALL phrase is used. *literal-2*, *identifier-2* or *literal-4*, *identifier-4* is substituted for each occurrence of *literal-1*, *identifier-1* or *literal-3*, *identifier-3*.
2. When the LEADING phrase is used, the substitution of *literal-2*, *identifier-2* or *literal-4*, *identifier-4* terminates as soon as a character other than *literal-1*, *identifier-1* or *literal-3*, *identifier-3* or the right-hand boundary of the data item is encountered.
3. When the UNTIL FIRST phrase is used, the substitution of *literal-2*, *identifier-2* or *literal-4*, *identifier-4* terminates as soon as *literal-1*, *identifier-1* or *literal-3*, *identifier-3* or the right-hand boundary of the data item is encountered.
4. When the FIRST phrase is used, the first occurrence of *literal-1*, *identifier-1* or *literal-3*, *identifier-3* is replaced by *literal-2*, *identifier-2* or *literal-4*, *identifier-4*.
5. When the TRAILING phrase is used, the substitution of *literal-2*, *identifier-2* or *literal-4*, *identifier-4* terminates as soon as a character other than *literal-1*, *identifier-1* or *literal-3*, *identifier-3* or the left-hand boundary of the data item is encountered.

When the APPLY INLINE EXAMINE clause is specified in the Inline Section of the CONTROL DIVISION, inline code will be generated for all EXAMINE statements except when the item being examined is a signed numeric.

The INSPECT statement offers many of the same features as EXAMINE plus several significant extensions which make INSPECT more desirable or appropriate than EXAMINE. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

2.4.4. The EXHIBIT Statement

Function

The EXHIBIT statement is a debugging tool which causes a formatted display of specified items to be written to the program print file.

Format

```
EXHIBIT  { NAMED         }  { data-name-1 } [ , { data-name-2 } ] ...
         { CHANGED       }  { literal-1   }     { literal-2   }
         { CHANGED NAMED }
```

Description

The *data-name* can be qualified. The *literal* may be any nonnumeric literal or any figurative constant except ALL. A figurative constant, if used, is interpreted as a single-character literal.

An EXHIBIT NAMED statement functions like a DISPLAY statement with the following string substituted for each *data-name* in the syntax:

```
identifier-name = data-name-value
```

with two to five spaces inserted before the symbol '=', a single space following the symbol '=', and one through four spaces inserted after the *data-name-value*.

The *data-name-value* is converted, if necessary, to a printable form. COMPUTATIONAL, COMPUTATIONAL-4, and exact binary items are converted to their decimal equivalent. COMPUTATIONAL-1 and COMPUTATIONAL-2 items are printed as 10- and 18-digit integers with decimal places truncated.

An EXHIBIT CHANGED statement is the same as an EXHIBIT NAMED statement except that for each *data-name* which has not changed value since the last execution of this EXHIBIT CHANGED statement, blanks are printed instead of the name and value of the item. An EXHIBIT CHANGED statement reveals, both by *data-name* and column position, which variables have changed value. If none have changed value, a blank line is printed.

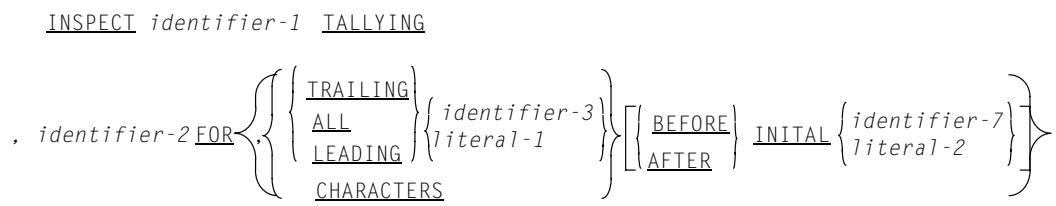
The EXHIBIT CHANGED NAMED statement is the same as the EXHIBIT CHANGED statement. except that changed values are printed consecutively without regard to column and blank lines are not printed.

When the total size of an EXHIBIT line exceeds the single transfer capacity of the printer or demand terminal, the EXHIBIT statement, like the DISPLAY statement, will cause multiple lines of output. Each EXHIBIT statement may list no more than 14 items if the

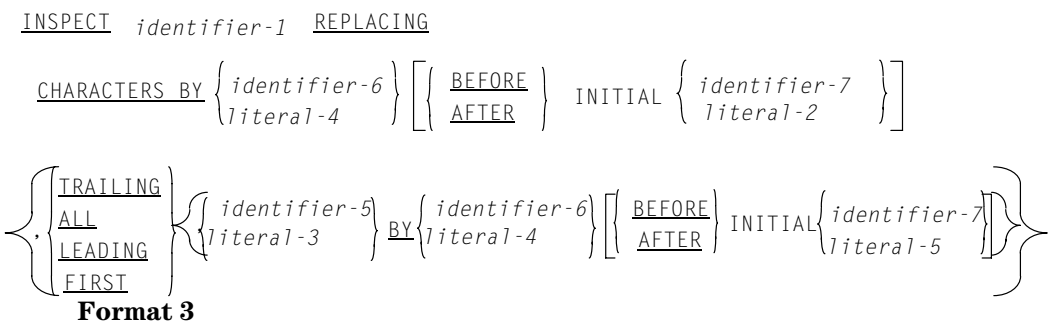
CHANGED phrase is used. and no more than nine items if the NAMED or CHANGED NAMED phrase is used.

2.4.5. The INSPECT Statement

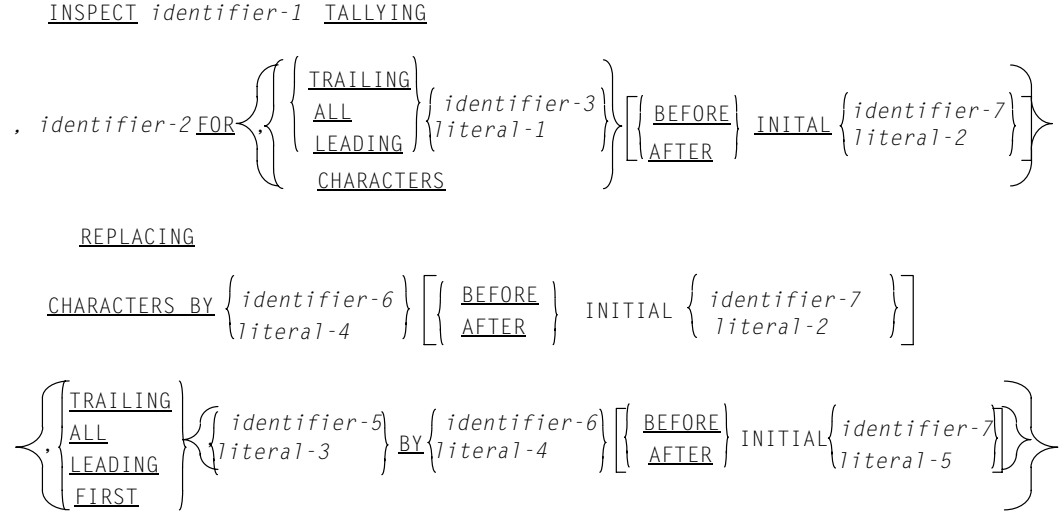
Format 1



Format 2



Format 3



Description

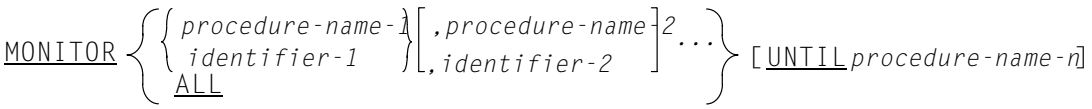
All rules for the LEADING phrase apply to the TRAILING phrase, except that the word “rightmost” must be substituted for the word “leftmost” and the words “last comparison cycle” must be substituted for the words “first comparison cycle” in all references to the LEADING phrase. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

2.4.6. The MONITOR Statement

Function

The MONITOR statement is a compiler-directing statement which causes the contents of a specified item to be written to the program print file each time the item is referenced.

Format



Description

The *identifier* may refer to any data item in the COBOL program. The parameter *identifier-1* must not be subscripted or indexed. If *identifier-1* is specified without subscripting or indexing and *identifier-1* refers to a table element. the effect is as if each element of the table had been individually specified. The label *procedure-name-1* may be any section or paragraph-name in the program. Both *identifiers* and *procedure-names* may be qualified as necessary.

The MONITOR statement can appear anywhere in the Procedure Division, but must be in a sentence by itself. Following the MONITOR sentence and continuing until either the source line containing the *procedure-name* label specified in the UNTIL phrase or the end of the program source is reached, the object program will be constructed so as to print out monitor information on each explicit reference to the identifiers or procedures specified in the MONITOR statement and on each definition of procedures specified in the MONITOR statement.

The ALL phrase means all data references and all procedure tags and references will be monitored over the specified range.

The monitor output line for data references is constructed after the execution of the statement containing the reference. When the referenced data is possibly changed as in arithmetic operations, moves. etc., monitor output is constructed both before and after the execution of the statement. For each reference, the line will contain a reference number which matches the reference number in the resolved source listing, the name of the *identifier*, and the value of the data item. Values are converted, as necessary, in the same manner as for the DISPLAY and EXHIBIT statements. A value of an index-name will be the occurrence number.

The monitor output for procedures is generated for both explicit transfers to the procedure and when execution “falls through” a specified procedure label.

The output line is similar to that for a data reference but does not contain a value or definition line number.

The selection of references for which to generate monitor calls is done statically at compile time, and not dynamically at object time. The compiler will not generate monitor code if the program is compiled with the M option on the @ACOB statement.

The inclusion of monitoring, particularly over a large range, will significantly increase the size of the resultant object program.

A user may dynamically control monitor output by explicitly setting and clearing the special register data item. (See 2.3.2.)

The MCFLAG field is implicitly defined as a 1-word binary integer initially set to 1. If MCFLAG is zero and a monitored identifier or procedure is referenced. the monitor output is suppressed. Therefore, by setting MCFLAG to zero and nonzero, the user can dynamically and selectively inhibit printing by the debugging monitor routine.

The Debug module of Standard COBOL 1974 has replaced the MONITOR statement. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

2.4.7. The NOTE Statement

Function

The NOTE statement allows the programmer to write commentary which is produced on the listing but not compiled.

Format

```
NOTE character-string.
```

Description

Any combination of the characters from the computer's character set may be included in the *character-string*.

If a NOTE statement is the first statement of a paragraph. the entire paragraph is considered to be part of the *character-string*. Proper format rules for paragraph structure must be observed.

If a NOTE statement appears as other than the first statement of a paragraph, the commentary ends with the first instance of a period followed by a space.

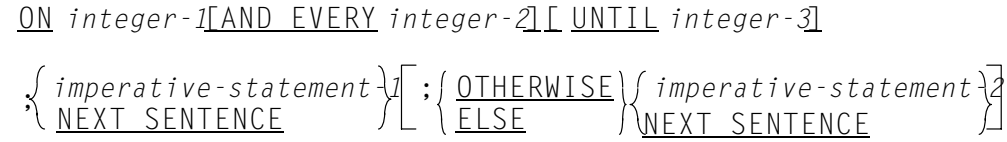
Standard COBOL 1974 has replaced the NOTE statement by a generalized comment facility. An asterisk (*) in character position 7 identifies any line as a comment line. A slash (/) in character position 7 causes the line to be treated as a comment and causes page ejection. The * and the /, unlike the NOTE, may appear anywhere in a COBOL program and do not in any way affect the compilation of subsequent statements. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

2.4.8. The ON Statement

Function

The ON statement is a conditional statement whose truth or falseness is determined by the number of times the statement has been executed.

Format



Description

A counter is created for this statement with an initial value of zero. The counter is automatically incremented by 1 immediately before each execution of the statement.

If *integer-2* and *integer-3* are absent (only *integer-1* is present), the ON condition is true once, when the counter equals *integer-1*, and not at any other time.

If *integer-2* is present and *integer-3* is absent, the ON condition is true whenever the counter equals *integer-1* + k* *integer-2*, where k is a positive integer.

If *integer-2* and *integer-3* are both present. the condition is true whenever the counter equals *integer-1* + k * *integer-2* and the counter is less than *integer-3*.

If integer2 is absent and *integer-3* is present, *integer-2* is assumed to be equal to 1.

If the ON statement appears in a USE FOR RANDOM PROCESSING declarative or in a subprogram. a separate counter is established for each processing cycle or for each time the subprogram is called or entered.

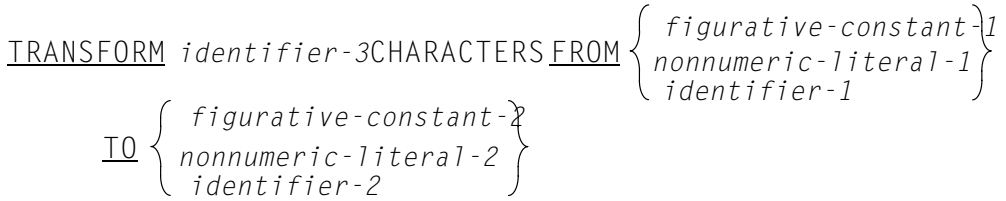
When the examination results in a true condition. the imperative statements preceding the OTHERWISE/ELSE are executed. When the condition is false, the imperative statements following OTHERWISE/ELSE are executed. In either case, control proceeds to the next sentence unless a GO TO directs execution elsewhere.

2.4.9. The TRANSFORM Statement

Function

The TRANSFORM statement is issued to alter characters according to a transformation rule.

Format



Description

The parameter *identifier-3* must represent an elementary alphabetic, alphanumeric, or numeric-edited item, or a group item.

The parameters *identifier-1* and *identifier-2* must be elementary alphabetic, or alphanumeric items, or fixed-length group items less than 97 characters in length. A fixed-length group is defined as a group which does not contain an OCCURS DEPENDING ON.

Literals and *figurative-constants* may not contain ALL.

A character may not be repeated in nonnumeric-*literal-1* or in the area defined by *identifier-1*. If a character is repeated, the results will be unpredictable.

Transformation, in general, results in a character in *identifier-3* being replaced if it is equal to any character in a FROM operand.

The replacement character is either:

- The TO operand character if the TO operand is a single-character item, or
- The character that is in the same relative position in the TO operand as the match character is in the FROM operand.

If the TO operand is not a single-character item, the TO and FROM operands must be of equal length. The figurative constant is by definition a single-character item.

The INSPECT statement offers many of the same features as the TRANSFORM statement plus several significant extensions which make the INSPECT statement more desirable or appropriate than the TRANSFORM statement. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

Section 3
Sequential I-O

3.1. Environment Division

3.1.1. The FILE-CONTROL Paragraph

3.1.1.1. The ASSIGN Clause

Format

```
ASSIGN TO implementor-name-1[ , implementor-name-2] ...
```

Description

Implementor-name-1 may be one of the following depending on the external media:

MASS-STORAGE	SEQUENTIAL-FILE
MASS-STORAGE-28	UNISERVO
MASS-STORAGE-56	UNISERVOS
MASS-STORAGE-112	770-PRINTER

In the forms, MASS-STORAGE-*nnn*, the integer value *nnn* refers to the prepping factor of the mass storage files. These alternatives are available to allow blocking of records to be sensitive to different physical record sizes on disk devices. The block size will be calculated according to the BLOCK CONTAINS clause and then rounded up to the next multiple of the specified physical record size. With the *implementor-name* MASS-STORAGE, a default value of 28 words is used in rounding up block size (unless the user changes FC$PREP to 56 or 112, then 56 or 112 is used).

For files assigned to other than 770-PRINTER:

- If *implementor-name-2* is specified, it is a 1- to 12-character name used to link the COBOL file to OS 1100 Exec file control and should appear on an @ASG or @USE statement in the run stream for executing the object program.
- When *implementor-name-2* is not specified, the first 12 characters of the file-name are used to link the COBOL file to the OS 1100 Exec file control.

For files assigned to 770-PRINTER:

- When present, *implementor-name-2* is a 1- to 12-character name specifying an alternate system output file.
- When *implementor-name-2* is not specified, *implementor-name-1* is linked to the system output files associated with each run unit.

All recursions of *implementor-name-1*, *implementor-name-2* are for documentation only.

3.1.1.2. The FILE-LIMITS Clause

Format

```
[ ;{FILE-LIMIT IS  }{ data-name-1 }{ THRU      }{ data-name-2 }
  {FILE-LIMITS ARE}{ literal-1   }{ THROUGH  }{ literal-2   }

    [ ;{ data-name-3 }{ THRU      }{ data-name-4 } ]... ]
       { literal-3   }{ THROUGH  }{ literal-4   }
```

Description

The FILE-LIMITS clause specifies that logical records are retrieved or placed sequentially in the file by the implicit progression from one segment to another. Each pair of operands associated with the keyword THRU represents a logical segment of the file. The logical beginning of the file is that address represented by the first operand of the first or only pair of the FILE-LIMITS clause. The logical end of the file is that address represented by the last operand of the last pair of the FILE-LIMITS clause.

The parameters *data-name-1*, *literal-1*, etc., must be positive numeric integers with no positions to the right of the assumed decimal point.

The parameters *data-name-1*, *literal-1*, etc., represent the relative record number of a within the file.

When a file containing a FILE-LIMITS clause is opened for output. the file handler ignores the BLOCK CONTAINS clause, and the file is created with one logical record per physical block.

When a file containing a FILE-LIMITS clause is opened for input or I-O, processing will be in error unless the file has been created with a FILE-LIMITS clause or with ORGANIZATION DIRECT.

All physical blocks are allocated the same amount of mass storage. This allows the Mass Storage Control System (MSCS) to make direct address FILE-LIMITS computations for operands.

The FILE-LIMITS clause may not be specified by RECORDING MODE SDF files.

3.1.1.3. The PROCESSING MODE Clause

Function

The PROCESSING MODE clause specifies the order in which logical records are processed by the COBOL program.

Format

```
[,PROCESSING MODE IS SEQUENTIAL ]
```

Description

The PROCESSING MODE clause specifies the order in which logical records are processed by the COBOL program.

When PROCESSING MODE IS SEQUENTIAL is specified, the logical records are normally processed in the order in which they are accessed. In this case, only one area is allocated to contain a logical record. If the COBOL program has reference to such a file in a USE FOR RANDOM PROCESSING declarative section, appropriate lock techniques must be used by the programmer to ensure the integrity of the logical record if simultaneous accesses to it are possible. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

3.1.1.4. The RESERVE Clause

Format

```
[ , RESERVE { integer-1 } ALTERNATE [ AREA  ] ]
            { NO        }          [ AREAS ]
```

Description

The RESERVE clause is used to specify the number of buffer storage areas used to be in processing the file.

For synchronously processed files; in other words, those for which PROCESSING MODE is SEQUENTIAL. a maximum of two buffer areas is allocated regardless of the number specified by *integer-1*. For input/output (I-O) files that are being randomly processed (in other words, contain the PROCESSING MODE IS RANDOM clause; see the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version)). *integer-1* buffer areas are allocated. If the RESERVE clause is not specified, two buffer areas are allocated. If NO is specified, only one buffer area is allocated.

Buffer areas are not compiled as part of the object program. Instead, during execution, the object program is dynamically extended to allow for the buffer areas when the file is opened. The buffer areas are released when the file is closed.

The RESERVE clause may not be specified for RECORDING MODE SDF files.

3.1.2. The I-O-CONTROL Paragraph

3.1.2.1. The APPLY WRITE-ONLY Clause

Format

```
[ ;APPLY WRITE ONLYON file-name-1[ ,file-name-2]...]
```

Description

The APPLY WRITE ONLY clause has no meaning for unit record or mass storage files. For tape files, it directs the compiler to allocate an area separate from the buffer area to contain the logical record. Such a file is referred to as a move mode file. A file for which the logical record is contained within the buffer area is referred to as a locate mode file.

The APPLY WRITE-ONLY clause can be used for files opened either for input or output. Its primary purpose however, is to allow variable length records to be packed into a block for output. Each record is moved into the buffer after it. has been processed and consequently requires minimum space. If the clause is absent, the records are processed directly in the buffer where each record of the file occupies the area required for the largest record of the file.

This clause (actually. packing of variable length records) is implied by other clauses referencing the same file. The other clauses are: SAME RECORD AREA, RECORDING MODE other than INTERNAL, and PROCESSING MODE IS RANDOM.

The APPLY WRITE-ONLY clause may not be specified for RECORDING MODE IS SDF files.

3.1.2.2. The RERUN Clause

Format

```
[ ; RERUN [ ON { file-name-1   } ] ]
              { UNISERVO       }
              { UNISERVOS      }
              { MASS-STORAGE   }

    EVERY { END OF      { REEL    } } OF file-name-2 ] ...
          { integer-1   { UNIT    } }
                        { RECORDS }
```

Description

The RERUN EVERY *integer* RECORDS format is the only one recognized by RECORDING MODE IS SDF files. Also, *file-name-1* of the RERUN clause cannot RECORDING be a MODE IS SDF file.

3.2. Data Division

3.2.1. The BLOCK CONTAINS Clause

Format

```
[ ; BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS    } ]
                                              { CHARACTERS }
```

Description

For RECORDING MODE IS SDF files, the BLOCK CONTAINS clause specifies the size of the physical block. Absence of this clause will cause block size to default to 224 words; when records are greater than 224 words in length, the blocks will be spanned. The presence of the clause will cause a block size to be calculated as follows:

- If the CHARACTERS phrase is specified, *integer-2* is divided by 4 and then rounded up to a 112-word multiple.
- If the RECORDS phrase is specified, *integer-2* is multiplied by the maximum record length in words plus 1 and then rounded up to a 112-word multiple.

3.2.2. The LABEL RECORDS Clause

Format

```
; LABEL { RECORD IS  } { STANDARD                                  }
       { RECORDS ARE } { OMITTED                                   }
                       { data-name-1 [,data-name-2] ...            }
```

Description

The LABEL RECORDS clause is required for each file description.

3.2.2.1. Tape Files

Label records are located at both the beginning and end of a file or clause reel. This permits the identification of these label records.

If the user desires only the system standard label records, then the STANDARD phrase may be used. The OMITTED phrase specifies that no explicit labels exist for the file or for the device to which the file is assigned.

OMITTED may not be specified on a tape file which is to be opened is REVERSED or referenced by the MULTIPLE FILE clause.

3.2.2.2. Mass Storage Files

Label records on a mass storage file always occupy sectors 0 through 3 of that file. The LABEL RECORDS clause permits the identification of these label records.

3.2.2.3. Tape and Mass Storage Files

The label record *data-name-1* must be the subject of a Record Description entry associated with the file. The *data-name* must not appear in the DATA RECORDS clause of the File Description.

When user label records are specified (*data-name-1*, *data-name-2*.. . .), the user may define fields in the description of *data-name-1* (up to 80 characters) and have values placed in them (OUTPUT) and checked for (INPUT and I-O) the by using VALUE OF clause. All fields described in a LABEL RECORDS clause must have DISPLAY usage. Such fields may not be referred to outside of the USE for LABEL declarative sections in the Procedure Division.

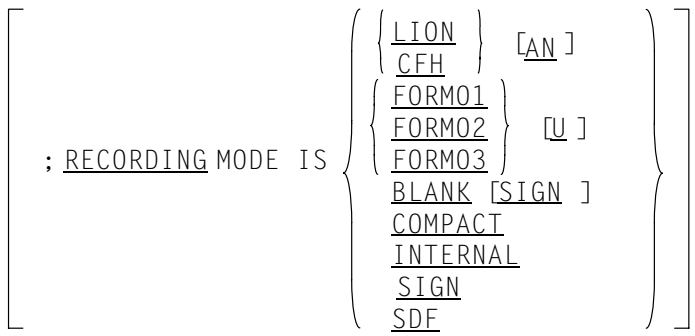
The specification of user labels indicates the presence of these labels in addition to the system standard labels.

3.2.2.4. RECORDING MODE SDF Files

RECORDING MODE IS SDF files have their own labeling conventions which are independent of conventional COBOL labels. Hence, the LABEL RECORDS clause is ignored for these files.

3.2.3. The RECORDING MODE Clause

Format



Description

The RECORDING MODE clause is not meaningful for unit record files or report files. For tape and mass storage files, it specifies the format of the logical records or record control information on tape or mass storage which may be different from the format in main storage. Any appropriate data conversions are performed as data records are transferred between main storage and tape or mass storage.

When RECORDING MODE is INTERNAL or SDF is specified, it means that all records are read or written exactly as they appear in main storage with no data conversion taking place. If a RECORDING MODE clause is not present. INTERNAL is assumed.

If AN, FORM01, FORM02 FORM03, SDF, or INTERNAL is not specified and the file contains multiple data record descriptions, then a record selector field within each record description is required. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version) for a description of record selector fields.

When the record is read or written, the selector is tested to determine which record conversion must be performed.

To guarantee a unique external representation of +0 and -0 for signed numeric fields on ASCII output files, the SIGN option must be specified.

If a REDEFINES clause is used within a record description, then the original definition is used rather than the redefinition for the purposes of record conversion.

3.2.3.1. The RECORDING MODE IS BLANK Clause

When the RECORDING MODE IS BLANK clause is stated, leading blanks in numeric DISPLAY or DISPLAY-1 fields are converted to leading zeroes as the read records are from tape or mass storage.

3.2.3.2. The RECORDING MODE IS BLANK SIGN Clause

The RECORDING MODE IS BLANK SIGN clause means that the conversions for both RECORDING MODE IS BLANK and RECORDING MODE IS SIGN are to be performed on the file.

3.2.3.3. The RECORDING MODE IS CFH Clause

The RECORDING MODE IS CFH clause indicates that any tape written or read the is in external format compatible with that produced by the COBOL File Handler (CFH).

When the AN option is used, the records are assumed to consist entirely of Fieldata characters. No data conversion is performed on these files. All record descriptions must be specified as USAGE DISPLAY-1 fieldata).

When the AN phrase is not used, the following conversion is performed when transferring a logical record between main storage and tape:

- USAGE IS COMPUTATIONAL-4. For all fields greater than 11 digits in size, the field has a single sign in main storage and it carries two signs on tape.
- CFH files must contain 28-word label blocks. Only the first four words of each block will be recognized and processed by ASCII COBOL. Neither VALUE OF nor label declarative processing is performed. Label blocks on output files contain only the minimum information necessary for proper processing. Files must use the LABEL RECORDS ARE OMITTED or STANDARD clause.

3.2.3.4. The RECORDING MODE IS COMPACT Clause

The RECORDING MODE IS COMPACT clause means that the records on the file contain variable length arrays. Each record on tape is compressed and is converted to or from expanded form in main storage. The DEPENDING phrase of the OCCURS clause is used whenever a variable number of occurrences of a dimensioned item is desired. If a DEPENDING phrase is used within a record description of a file whose RECORDING MODE IS COMPACT, the value of a data item is used to specify the conversion required so that unused occurrences do not appear in the external media.

If a RECORDING MODE IS COMPACT file specifies the APPLY EXDEF clause and the file contains a forward-referenced OCCURS DEPENDING ON data item, the APPLY EXDEF clause can only be used in the main program.

3.2.3.5. The RECORDING MODE IS FORM01, FORM02, or FORM03 Clause

The RECORDING MODE IS FORM0X clause specifies that the tape was written to or is to be read by a system which processes tapes with this external format.

All record descriptions must be specified as USAGE DISPLAY-1 (Fieldata).

When the U option is not used, the last block is padded with records of all 9's when necessary to fill the block. These records are given to the user as valid records on input.

When the U option is used, the last block will not be padded to fill a short block.

Compatible files may contain label blocks, but neither VALUE OF nor label declarative processing is performed.

3.2.3.6. The RECORDING MODE IS LION Clause

The RECORDING MODE IS LION clause specifies that the tape was written or be is to read by a system which processes tapes with this external format.

This format is different from the standard tape format used by the ASCII COBOL compiler and is included for compatibility and interchangeability with other systems.

When the AN phrase is used, the records are assumed to consist entirely of Fieldata characters. No data conversion is performed on these files. All record descriptions must be specified as USAGE DISPLAY-1 (Fieldata).

When the AN phrase is not used, the following conversions will be made when transferring a logical record to and from main storage and tape:

- USAGE IS DISPLAY-1 (Fieldata). For signed numeric fields, the sign is carried in main storage as an overpunched sign on the lower order digit. On tape, the sign is carried as a 6-bit byte preceding the high order digit of the field.
- USAGE IS COMPUTATIONAL-4 (Fieldata). For all fields greater than 11 digits in size, the field has a single sign in main storage and it carries two signs on tape.

LION files can be of two forms: fixed length record format and variable length record format. The variable length record format is selected when two or more record descriptions with different lengths are present for the file.

LION files may contain label blocks, but neither VALUE OF nor LABEL declarative processing is performed Label blocks are written to LION output files containing the, minimum information necessary for the files to be processed properly. Files must use the LABEL RECORDS ARE OMITTED or STANDARD clause.

3.2.3.7. The RECORDING MODE IS SDF Clause

The RECORDING MODE IS SDF clause specifies that the file is in the System the Data Format (SDF] on external media. These files are processed by the OS 1100 Processor Common Input/Output System (PClOS).

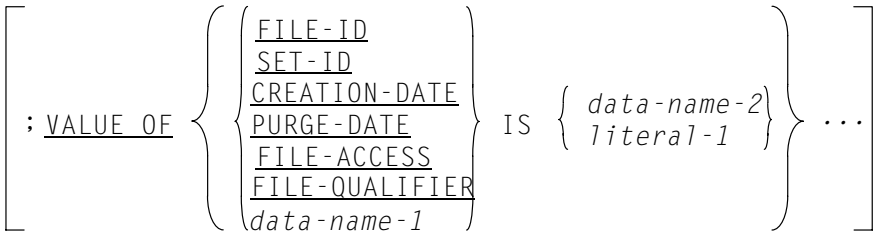
For Sequential Organization files assigned to MASS-STORAGE, this file format format is identical to the file used for Sequential Organization files assigned to DISC. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

3.2.3.8. The RECORDING MODE IS SIGN Clause

The RECORDING MODE IS SIGN clause indicates that every signed numeric data DISPLAY or DISPLAY-1 item is given an overpunched sign on the low order digit as it is read into main storage. On mass storage or tape the sign is carried as an explicit character preceding the high order digit of the field.

3.2.4. The VALUE OF Clause

Format



Description

The VALUE OF clause specifies the value of an item in a label record. This clause should not be present for a file when LABEL RECORDS ARE OMITTED is specified.

Since files with RECORDING MODE IS LION, CFH, SDF, FORM01, FORM02, or FORM03 do not have system standard labels, this clause has no meaning and is ignored.

The following are the names of fields in the system standard labels and have the implied definition of:

	Field Name	Implied Definition
02	FILE-ID	PIC X(17)
02	SET-ID	PIC X(6)
02	CREATION-ID	PIC 9(6)
02	PURGE-DATE	PIC 9(6)
02	FILE-ACCESS	PIC X(1)
02	FILE-QUALIFIER	PIC X(12)

In the absence of any or all of these names in the VALUE OF clause when a file is opened for output, the following values will be assumed:

- For tape file sets using the 8-bit tape labeling system:
  - FILE-ID

    will be set to *qualifier*external-name*. The *qualifier* will be truncated if the FILE-ID is longer than 17 characters.
  - SET-ID

    will be set to the first reel number.
  - FILE-QUALIFIER

    will be set to "U1100-1".
- For all other file types:
  - FILE-ID

    will be set to the 12-character external file name.
  - SET-ID

    will be set to the first reel number, except when the file is initially opened on a nonfirst reel it will be set to "UNIVAC".
  - CREATION-DATE

    will be set to the current date.
  - PURGE-DATE

    will be set to the current date plus the expiration period specified on the @ASG statement or system default if no expiration is specified.
  - FILE-ACCESS

    will be set to a space indicating unlimited access to the file.
  - FILE-QUALIFIER

    will be set to the qualifier of the file.

With the 8-bit tape labeling system, the CREATION-DATE cannot be set by the user.

In addition to these system standard label names, the VALUE OF *data-name-1* form may be used provided LABEL RECORDS ARE data-name is also specified. In this case, *data-name-1*. etc., must be defined in the record description for a label record.

The contents of *data-name-2* or *literal-1* will be moved to the label field on output or checked against the label field on input. The parameter *data-name-2* must be described as USAGE DISPLAY (ASCII).

When user label records are specified, the VALUE OF processing takes place in two steps:

1. When the system standard label records are being read or written, the VALUE OF standard label names are checked or moved, respectively.
2. When user label records are being read or written, the VALUE OF data-name clauses are checked or moved, respectively.

3.2.4.1. The VALUE OF CREATION-DATE Clause

The contents of this field are set to the current date at the time the file is opened for output. The VALUE OF CREATION-DATE clause modifies the contents of the field to the value specified. The value field is written in the COBOL program in the form *MMDDYY* and is converted to Julian form when the HDR1 label is created. At this time, the contents of the month, day, and year values are verified for a legal range of numbers. With the 8-bit tape labeling system, this field cannot be set by the user.

3.2.4.2. The VALUE OF *data-name* Clause

In addition to the system standard labels, tape and mass storage files may optionally include user labels.

In order to create user labels on an output file. the COBOL program must include the LABELS ARE *data-name* clause. User labels are then created by the use of the VALUE OF *data-name* clause or by the specification of one or more USE ... LABEL PROCEDURE declarative sections.

The VALUE OF processing is performed each time a user label may be created. No distinction is made between references to data names in different label record descriptions. It is the users responsibility to ensure that the VALUE OF items are meaningful at the time a user label may be created.

This chart indicates the time at which a user label may be created (or processed on input). and the sequence of processing:

Label Identifier	Events Causing Label Processing to Occur	Sequence of Label Processing
UVL-User beginning volume labels (9-bit tape labeling only)	• OPEN • CLOSE REEL (after tape swap) • End of reel on READ or WRITE	1. BEFORE BEGINNING REEL declarative is executed. 2. VALUE OF processing. 3. AFTER BEGINNING REEL declarative is executed.

continued

Label Identifier	Events Causing Label Processing to Occur	Sequence of Label Processing
UHL-User file header labels	OPEN	1. BEFORE BEGINNING FILE declarative is executed. 2. VALUE OF processing. 3. AFTER BEGINNING FILE declarative is executed.
UTL-User end-of-file labels	CLOSE	1. BEFORE ENDING FILE declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING FILE declarative is executed.
UTL-User end-of-volume labels	• End of reel on CLOSE file • CLOSE REEL on output file • End of reel on READ or WRITE	1. BEFORE ENDING REEL declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING REEL declarative is executed.

For output files. the user label is written only if the label identifier is set correctly in the label record area. When a valid label exists, it is written.

For input files, label processing is performed only if a label with that appropriate identifier exists.

3.2.4.3. The VALUE OF FILE-ACCESS Clause

The VALUE OF FILE-ACCESS clause only has significance for 8-bit labeled tapes. The only allowable functioning characters are ASCII “1” through “7” and space (octal values 061 through 067,000, and 040, respectively). Any other character will be treated as a space (in other words, unrestricted access).

The function of the values 061-067 corresponds to the F, R, and W options on the COASG statement; e.g., if the R option is specified on the @ASG statement when the file is created, the file is read-only until it expires.

NUL indicates that the @ASG statement options F, R. and W will be used. This is equivalent to not specifying a FILE-ACCESS clause.

Accessibility Character	Assign Options		
	F	R	W
" " or 040	X		
"1" or 061			
"2" or 062	X		X
"3" or 063			X
"4" or 064	X	X	
"5" or 065		X	
"6" or 066	X	X	X
"7" or 067		X	X
NUL or 000	As specified on @ASG statement	As specified on @ASG statement	As specified on @ASG statement

3.2.4.4. The VALUE OF FILE-ID Clause

The file identifier is checked by the Exec each time a reel specified on an @ASG statement is mounted. It must match the file name specified on the @ASG statement unless the F option was present at creation time. In either case, the expiration date must indicate that the previous use of the file has expired.

If, when creating a file. the VALUE OF clause is used to modify the file identifier to some name other than the one specified as the external file name in the SELECT clause. that file may not be assigned in another run unless the file name on the @ASG statement is changed. The @USE statement may, however, be later used to equate the file identifier and the @ASG file name with the external file name used in the COBOL program.

For example, given a COBOL program which creates a file with the following clauses:

```
    .
    .
    .
SELECT MASTER-FILE ASSIGN TO UNISERVO XN
    .
    .
    .
VALUE OF FILE-ID IS "TAPE-FILE"
```

the execution of the program which creates the file (unless the @USE statement is used) will be preceded by an @ASG XN,T,*reel-no*.

Once the tape is created, if it is desired to read the file in another run using the same ASSIGN clause, the following procedure may be used:

```
@ASG  TAPE-FILE, T, reel-no
@USE  XN, TAPE-FILE
```

If the user wishes to subsequently access a labeled tape file without a VALUE OF FILE-ID clause (in other words, non-COBOL use of the file), the qualifier and an asterisk (*) should be included in the FILE-ID value for use with an 8-bit labeled tape and should not be included in the FILE-ID value for use with a 9-bit labeled tape.

3.2.4.5. The VALUE OF FILE-QUALIFIER Cause

The contents of the FILE-QUALIFIER field are set originally to the qualifier specified in the @ASG statement or the qualifier specified in the currently active @QUAL statement (or by default to the project field in the @RUN statement). The VALUE OF FILE-QUALIFIER clause modifies the contents of the field to the value specified. As was noted for the file identifier, this field is checked by the Exec each time the reel is assigned to a run. The qualifier on the ASG statement must agree with the qualifier in the HDR1 label unless the purge date shows the file has expired.

With the 8-bit tape labeling system, this field cannot be set by the user.

3.2.4.6. The VALUE OF PURGE-DATE Clause

The contents of the PURGE-DATE field are set to the current date at the time the file is opened for output plus the number of days to expire specified in the @ASG statement or system default if not specified in the @ASG statement. The VALUE OF PURGE-DATE clause modifies the contents of the field to the value specified. The rules for the value for PURGE-DATE are the same as for CREATION-DATE.

The contents of the PURGE DATE field represent the day on which the file may be overwritten. The Exec verifies each reel assigned to a run by checking if the file identifier and qualifier agree with the @ASG statement and the current date is the same or later than the purge date in the HDR1 label.

3.2.4.7. The VALUE OF SET-ID Clause

The COBOL program may set the contents of this field without restriction.

3.3. Procedure Division

3.3.1. The CLOSE Statement

Format

```
CLOSE file-name-1  [ REEL ]  [ WITH { NO REWIND } ]
                   [ UNIT ]         { LOCK      }

   [ ,file-name-2  [ REEL ]  [ WITH { NO REWIND } ] ] ...
                   [ UNIT ]         { LOCK      }
```

Description

The CLOSE verb terminates the processing of one or more input or one or more output files or reels and provides optional rewinding or locking. Each *file-name* refers to an FD description in the Data Division. An OPEN statement must be executed prior to the CLOSE statement.

The CLOSE *file-name* option, as applied to the entire file rather than to individual reels, initiates the final closing conventions for the file and releases the data area. A file may be closed once, but not more than once, for each time the file is opened. The UNIT or REEL phrase has no meaning for mass storage files.

For an output file, the final closing conventions for the file are performed and the data area is released. Furthermore, for either an input or an output tape file:

- If neither the LOCK nor NO REWIND phrase is specified, the current reel of the file is rewound and all other reels belonging to the file are rewound. However, this rule does not apply to those reels controlled by a prior CLOSE REEL entry.
- If the NO REWIND phrase is specified, the current reel of the file remains in whatever position it is in at the time the CLOSE is given.
- The REEL and UNIT phrases are not allowed for RECORDING MODE SDF files.

If the LOCK phrase is specified for a tape file, all reels belonging to the file are rewound with interlock except for those reels controlled by a prior CLOSE REEL. If the LOCK phrase is specified for a mass storage file, the file can never be reopened in the current execution.

The CLOSE *file-name* REEL option may be used for input or output files. The LOCK option may be used and the current reel will be rewound with interlock. The necessary processing is performed.

When a CLOSE REEL option is given, the locking or rewinding options of CLOSE REEL, if used, take precedence for the current reel and only the current reel, regardless of the options associated with a CLOSE of file. When a CLOSE *file-name* is given, its options are executed wherever possible for all mounted reels of the file except for those reels which may have been closed by a CLOSE REEL whose locking and rewinding options differ from those of the CLOSE *file-name*.

If the file has been specified as OPTIONAL and is not present, the standard end-of-file processing is bypassed.

For multiple reel files, the opening and closing of individual reels is automatic. However, the programmer must close the file when processing is to be terminated.

A CLOSE *file-name-1* clause should be executed for each file that was opened.

3.3.1.1. Tape Files

For labeled tape files. the CLOSE REEL label procedures are the same as those described for end-of-reel conditions on READ and WRITE, with the exception that there is no label processing performed for the current reel of an input file.

For labeled tape files (except LION, CFH, SDF, FORM01, FORM02, and FORM03), a CLOSE of the file causes the following label processing to occur:

- Input file, LABEL RECORDS ARE STANDARD
  1. If the end-of-file condition has been reached, steps 2 through 4 are performed. Otherwise, no label processing is done.
  2. The BEFORE ENDING FILE declarative is executed.
  3. The VALUE OF items are verified against the EOF1 label.
  4. The AFTER ENDING FILE declarative is executed.
- Input file, LABEL RECORDS ARE *data-name*
  1. If the end-of-file condition has been reached, steps 2 and 3 are performed. Otherwise, no label processing is done.
  2. The EOF label is verified against the VALUE OF items referencing standard label names.
  3. For each UTL label which is read from the tape the following occurs:
     a. The BEFORE ENDING FILE declarative is executed.
     b. The VALUE OF items referencing user label names are verified.
     c. The AFTER ENDING FILE declarative is executed.
- Output file, LABEL RECORDS ARE STANDARD
  1. A hardware EOF mark is written to tape.
  2. An EOF1 label record is created with standard default values.
  3. The BEFORE ENDING FILE declarative is executed.
  4. The VALUE OF items are moved to the label record.
  5. The AFTER ENDING FILE declarative is executed.
  6. The EOF1 and EOF2 label records are written to tape followed by two hardware EOF marks.

- Output file, LABEL RECORDS ARE *data-name*
  1. A hardware EOF mark is written to tape.
  2. An EOF1 label record is created with the standard default values.
  3. The VALUE OF items referencing standard label names are moved to the label record.
  4. The EOF1 label record is written to tape followed by the EOF2 label.
  5. The label record. character positions $ through 80, is cleared to spaces.
  6. The BEFORE ENDING FILE declarative is executed.
  7. The VALVE OF items referencing user label names are moved to the label record.
  8. The AFTER ENDING FILE declarative is executed.
  9. The first three characters of the label record aye examined. If they contain the characters UTL, the label is written to tape and the processing is repeated at step 5. If the first three characters of the label area do not contain UTL, or the first four characters remain unchanged after the reiteration process, the label record is not written to tape. Finally, two hardware EOF marks are written to tape.

For tape files using the 8-bit tape labeling system, writing a tape mark is an integral part of writing the end-of-file labels and standard label items cannot be changed in these steps.

For LION, CFH, FORM02, and FORM03 files, an end-of-file label block is written to tape followed by two hardware EOF marks. For FORM01 files, two hardware EOF marks are written.

3.3.1.2. Mass Storage Files

For labeled mass storage files (except SDF), closing through a CLOSE statement causes the following label processing to occur:

- Input or I-O file, LABEL RECORDS ARE STANDARD
  1. The EOF1 label is read.
  2. The BEFORE ENDING FILE declarative is executed.
  3. The VALUE OF items are verified against the EOF1 label.
  4. The AFTER ENDING FILE declarative is executed.
  5. If the file is I-O, the label record is rewritten.

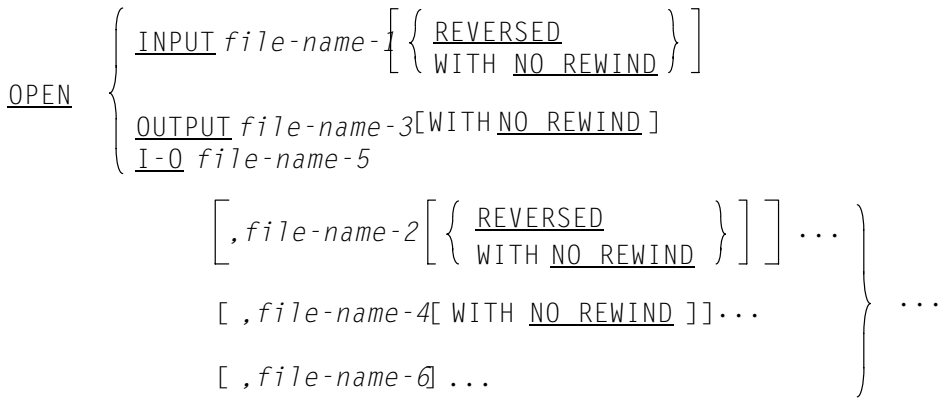
- Input or I-0 file, LABEL RECORDS ARE *data-name*
  1. The EOF label is read and verified against the VALUE OF items referencing standard label names.
  2. If a UTL label is present, the following occurs after it is read from the mass storage file:
     a. The BEFORE ENDING FILE declarative is executed.
     b. The VALUE OF items referencing user label names are verified.
     c. The AFTER ENDING FILE declarative is executed.
     d. If the file is I-O, the label record is rewritten.
- Output file, LABEL RECORDS ARE STANDARD
  1. An EOF1 label record is created with standard default values.
  2. The BEFORE ENDING FILE declarative is executed.
  3. The VALUE OF items are moved to the label record.
  4. The AFTER ENDING FILE declarative is executed.
  5. The EOF1 label record is written.
- Output file, LABEL RECORDS ARE *data-name*
  1. An EOF1 label record is created with the standard default values.
  2. The VALUE OF items referencing standard label names are moved to the label record.
  3. The EOF 1 label record is written.
  4. The label record, character positions 5 through 80, is cleared to spaces.
  5. The BEFORE ENDING FILE declarative is executed.
  6. The VALUE OF items referencing user label names are moved to the label record.
  7. The AFTER ENDING FILE declarative is executed.
  8. The first three characters of the label record are examined. If they contain the characters UTL, the label is written. If the first three characters of the label area do not contain UTL, the label record is not written.

3.3.1.3. RECORDING MODE SDF Files

For RECORDING MODE is SDF files, the end-of-file labels are processed by the Processor Common Input/Output System (PClOS). No declaratives are executed for these files.

3.3.2. The OPEN Statement

Format



Description

The OPEN statement initiates processing of the named files by checking and writing labels and performing any other input/output operations necessary prior to accessing the first record in a given file. However, the OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed.

If an input file is designated as OPTIONAL in the FILE-CONTROL paragraph, the Mass Storage Control System interrogates for the presence of the file. If the file is not present, the first READ statement for the file causes the imperative statement in the AT END phrase to be executed.

The REVERSED option may not be used on a file whose recording mode is LION, CFH, SDF, FORM01, FORM02, or FORM03 or on a file assigned to mass storage.

When using the 9-bit tape labeling system and the REVERSED option is specified, the MSCS will interrogate the position of the tape to ensure that it is set to read the last physical block on the reel. No label processing is performed when the file is opened REVERSED.

Tape file sets using the 8-bit tape labeling system can check labels when opened REVERSED and can be positioned using the MULTIPLE FILE clause.

The NO REWIND option is used to maintain position on a tape reel containing multiple files. It may be used when a file contained on a multiple file tape is being created. Although it is not necessary on a standard- tape input file since the MSCS will provide automatic positioning. it does save I-O access time. When attempting to read a multifile tape whose recording mode is LION, CFH, SDF, FORM01, FORM02, FORM03, F(EBCDIC), V, or U(EBCDIC), this option is required since the file handler does no automatic positioning.

3.3.2.1. Tape Files

In the following description of label processing, it should be noted that REEL processing occurs only on the first file of a multiple file tape. Tape file sets using the 8-bit tape labeling system have reel processing at the beginning and end of each reel, although under this system, User Volume Header labels cannot be processed.

For tape files (except LION, CFH, SDF, FORM01, FORM02, FORM03, and files with LABELS ARE OMITTED), the following label processing occurs.

Open Output

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. If the file being opened is not the first file on the tape, processing begins at step 4.
  2. The BEFORE BEGINNING REEL declarative is executed.
  3. The AFTER BEGINNING REEL declarative is executed.
  4. The HDR1 label is created with standard values.
  5. The BEFORE BEGINNING FILE declarative is executed.
  6. The VALUE OF items referencing the standard label are moved to the label area.
  7. The AFTER BEGINNING FILE declarative is executed.
  8. The HDR1 label record is written to tape followed by the HDR2 label record followed by one hardware EOF mark.
- If the LABEL RECORDS ARE *data-name* clause is present:
  1. If the file being opened is not the first file on the tape, processing begins at step 7.
  2. The label record, character positions 5 through 80, is cleared to spaces.
  3. The BEFORE BEGINNING REEL declarative is executed.
  4. VALUE of *data-name* items are moved to the label record.
  5. The AFTER BEGINNING REEL declarative is executed.
  6. The first three positions of the label area are examined as follows:
     a. If they contain the characters UVL, the label record is written to tape. Then processing is repeated starting at step 2.
     b. If they do not contain the characters UVL, or the first four characters remain unchanged after the reiterative process, the REEL processing is complete.
  7. The HDR1 is created with the standard default values.
  8. The VALUE OF items referencing the standard label are moved to the label area. The HDR1 label is written followed by the HDR2 label.
  9. The label record, character positions 5 through 80, is cleared to spaces.
  10. The BEFORE BEGINNING FILE declarative is executed.
  11. VALUE OF *data-name* items are moved to the label record.

12. The AFTER BEGINNING FILE declarative is executed.
13. The first three positions of the label area are examined as follows:
    a. If they contain the characters UHL, the label record is written to tape. Then processing is repeated starting at step 9.
    b. If they do not contain the characters UHL, or the first four characters remain unchanged after the reiterative process, FILE processing is complete.
14. Finally, one hardware EOF mark is written to tape.

For tape file sets using the 8-bit tape labeling system, step 6 is not performed and step 14 is performed by the system.

Open Input

- If the LABEL RECORDS ARE STANDARD clause is present:

  1. If the file being opened is not the first file on the tape, processing begins at step 4.
  2. The BEFORE BEGINNING REEL declarative is executed.
  3. The AFTER BEGINNING REEL declarative is executed.
  4. The HDR1 label record is read (any UVL label records are bypassed).
  5. The BEFORE BEGINNING FILE declarative is executed.
  6. The VALUE OF items are verified.
  7. The AFTER BEGINNING FILE declarative is executed.
  8. The file is positioned beyond the hardware EOF mark following the label records.

  For tape file sets using the 8-bit tape labeling system, steps 4 and 5 are reversed and step 6 is performed by the system as the HDR1 label is read.

- If the LABEL RECORDS ARE *data-name* clause is present:

  1. If the file being opened is not the first file on the tape, processing begins at step 3.
  2. For each UVL label record preceding the HDR1, the following process occurs:
     a. The BEFORE BEGINNING REEL declarative is executed.
     b. The VALUE OF items referencing user label items are verified.
     c. The AFTER BEGINNING REEL declarative is executed.
  3. The HDR1 label is read. The VALUE OF items referencing the standard label are verified.
  4. For each UHL label record following the HDR2 the following process occurs:
     a. The BEFORE BEGINNING FILE declarative is executed.
     b. The VALUE OF items referencing user label items are verified.
     c. The AFTER BEGINNING FILE declarative is executed.

5. Label processing is complete when the hardware EOF mark following the label records is reached.

For tape file sets using the 8-bit tape labeling system, step 2 is not performed. However, the BEFORE BEGINNING REEL and AFTER BEGINNING REEL declaratives will each be executed once.

3.3.2.2. Mass Storage Files

For mass storage files (except SDF files or files with LABEL RECORDS ARE OMITTED), the following label processing occurs.

Open Output

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. The HDR1 label is created with standard values.
  2. The BEFORE BEGINNING FILE declarative is executed.
  3. The VALUE OF processing is performed.
  4. The AFTER BEGINNING FILE declarative is executed.
  5. The HDR1 label record is written.
- If the LABEL RECORDS ARE *data-name* clause is present:
  1. The HDR1 label is created with standard values.
  2. The VALUE OF items referencing the standard label are moved to the label area.
  3. The HDR1 label is written.
  4. The label area, character positions 5 through 80, is cleared to spaces.
  5. The BEFORE BEGINNING FILE declarative is executed.
  6. VALUE OF *data-name* items are moved to the label record.
  7. The AFTER BEGINNING FILE declarative is executed.
  8. The user label is written if the label identifier (character positions 1 through 3) equals UHL.

Open Input or I-O

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. The HDR1 label record is read.
  2. The BEFORE BEGINNING FILE declarative is executed.
  3. The VALUE OF items are verified.
  4. The AFTER BEGINNING FILE declarative is executed.
  5. The EOF 1 label record is read.
  6. The file is positioned at the beginning of the data blocks.

- If the LABEL RECORDS ARE *data-name* clause is present:
  1. The HDR1 label is read. The VALUE OF items referencing the standard label are verified.
  2. If a UHL label is present following the HDR1 label, the following process occurs:
     a. The BEFORE BEGINNING FILE declarative is executed.
     b. The VALUE OF items referencing user label items are verified.
     c. The AFTER BEGINNING FILE declarative is executed.
     d. The UHL label is rewritten.
  3. The EOF1 label record is read.
  4. The file is positioned at the beginning of the data blocks.

3.3.2.3. RECORDING MODE SDF Files

For RECORDING MODE IS SDF files, the beginning-of-file labels are processed by the Processor Common Input/Output System (PClOS). No declaratives are executed for these files.

3.3.3. The READ Statement

Format

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

Description

For files designated as OPTIONAL in the FILE-CONTROL paragraph, the first execution of a READ statement will cause the AT END *imperative-statement* to be executed if the file is not assigned to the run.

For standard labeled tape files, the AT END is reached when a hardware EOF mark is read from tape followed by the EOF label record.

For unlabeled tape files, the AT END is reached when a hardware EOF mark is read from the last reel specified on the @ASG statement.

For LION, CFH, FORM02, and FORM03 tape files, the AT END is reached when an end-of-file label block is read.

For SDF files, the AT END is reached when an end-of-file label record is read.

For FORM01 tape files. it is not possible to tell whether an EOF or EOR is intended when two hardware marks are encountered. In this case, the AT END is reached when the reel count is equal to the number of reels specified in the @ASG statement. The reel count is checked against the number of reels specified on the @ASG statement. If they are equal, EOF is taken.

For mass storage files (except SDF), the AT END marker is reached when the current address of the record is equal to the end-of-file address established when the file was created, or if file limits were specified, the AT END is reached at the logical end of the last segment of the file and an attempt is made to read that file.

When the INTO option is used, the next logical record is made available in both the data area associated with the *identifier* and the input record area.

If the file is being processed in locate mode, the input record area is the location of the logical record in the buffer area (see 3.1.2.1).

For files in move mode, the logical record is moved from the buffer area to the logical record area. Any necessary data conversions are performed as part of the moving operation (see 3.2.3). All mass storage files are in move mode.

When a file consists of more than one type of record, a READ delivers the next record regardless of type; stated differently, all records of a given file share the main storage area. Thus, if there is more than one 01 entry in a given FD, it is the programmer's responsibility to determine which record is present at any particular instant.

Tape Files

When an end-of-reel is reached on a labeled tape file (except LION, CFH, SDF, FORM01, FORM02, and FORM03), the following label processing occurs:

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. The BEFORE ENDING REEL declarative is executed.
  2. The VALUE OF items are verified.
  3. The AFTER ENDING REEL declarative is executed.
  4. A tape reel swap is performed.
  5. The BEFORE BEGINNING REEL declarative is executed.
  6. The AFTER BEGINNING REEL declarative is executed.
  7. The tape is positioned beyond the hardware EOF mark and the first data block is read.

  For tape file sets using the 8-bit tape labeling system. system verification of the HDR1 label takes place after step 6.
- If the LABEL RECORDS ARE data-name clause is present:
  1. The EOF label is verified against the VALUE OF items referencing standard label names.
  2. For each UTL label following the EOF label the following occurs:
     a. The BEFORE ENDING REEL declarative is executed.
     b. The VALUE OF items referencing user label names are verified.
     c. The AFTER ENDING REEL declarative is executed.

3. When the hardware EOF mark is reached following the label blocks, a tape reel swap is performed.
4. For each UVL label record on the new reel, the following occurs:
   a. The BEFORE BEGINNING REEL declarative is executed.
   b. The VALUE OF items referencing user label names are verified.
   c. The AFTER BEGINNING REEL declarative is executed.
5. When the HDR1 label is read. the tape is positioned beyond the hardware EOF mark and the first data is read.

For tape file sets using the 8-bit tape labeling system, step 4 is not performed although the BEFORE BEGINNING REEL and AFTER BEGINNING REEL declaratives are each executed once. For 8-bit labeled tapes, system verification of the label takes place in Step 5.

3.3.4. The REWRITE Statement

Format

```
REWRITE record-name [ FROM identifier]

    ; INVALID KEY imperative-statement
```

Description

The REWRITE statement is used to replace a logical record on the file with a specified record. It is only valid for a file opened for I-O.

The record specified must not be longer than the original record it is replacing.

The REWRITE statement, when executed, performs all of the functions of a WRITE statement that follows a READ.

3.3.5. The USE Statement

Format 1

```
USE AFTER STANDARD ERROR PROCEDURE ON

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

Format 2

```
USE  { BEFORE }  STANDARD [ BEGINNING ]  [ REEL ]  LABEL PROCEDUREON
     { AFTER  }           [ ENDING    ]  [ FILE ]

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

Description

The USE statement specifies 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.

If the key words BEGINNING or ENDING are omitted, the designated procedures are executed for both beginning and ending labels.

If the key words FILE or REEL are omitted, the designated procedures are executed for both conditions. FILE declaratives are only executed once for BEGINNING (if specified) and once for ENDING (if specified) regardless of the number of reels on which the file is contained. REEL declaratives are executed for the first and all succeeding reels composing the file.

Declarative procedures are not executed for a file whose RECORDING MODE is LION, CFH, SDF, FORM01, FORM02, or FORM03.

The designated procedures are executed by the COBOL file handler at the appropriate time as follows:

- Format 1 is executed when an unrecoverable hardware error has occurred. Normal Exec recovery techniques were performed. The object program will be abnormally terminated after the declarative procedure has been executed.
- On tape files, Format 2 is executed on all OPEN and CLOSE statements and on READ and WRITE statements causing a tape reel swap. On mass storage files, Format 2 is executed on OPEN and CLOSE statements. The declarative is executed in conjunction with a specific standard or user label.

3.3.5.1. Tape and Mass Storage Files

- Input or I-O file - LABEL RECORDS ARE STANDARD
  1. Read label.
  2. Execute BEFORE declaratives.
  3. Verify label according to the VALUE OF clause.
  4. Execute AFTER declaratives.
- Output file - LABEL RECORDS ARE STANDARD
  1. Create skeleton label with default values.
  2. Execute BEFORE declaratives.
  3. Move data to label area according to VALUE OF clause.
  4. Execute AFTER declaratives.

3.3.5.2. Tape Files

- Input File - LABEL RECORDS ARE *data-name*
  1. Read a user label.
  2. Execute BEFORE declarative.
  3. Verify user label fields according to the VALUE OF clause.
  4. Execute AFTER declarative.
  5. Steps 1 through 4 are repeated until all user labels have been read.
- Output File - LABEL RECORDS ARE *data-name*
  1. Move spaces to character positions 5 through 80 in the label record area.
  2. Execute BEFORE declarative.
  3. Move data to user label fields according to the VALUE OF clause.
  4. Execute AFTER declarative.
  5. Examine label identifier to determine the following:
     a. If a proper label was created, write the label: move spaces to positions 5 through 80 of the label record area and repeat the process starting at step 2.

        When the process has been repeated, if the label identifier remains unchanged, no label is written, and label processing is complete.

     b. If the label identifier is not appropriate to the condition (e.g., identifier set to UTL1 on a beginning condition), no label is written and label processing is complete.

3.3.5.3. Mass Storage Files

- Input or I-O File - LABEL RECORDS ARE *data-name*
  1. Read user label.
  2. Execute BEFORE declaratives.
  3. Verify label according to the VALUE OF clause.
  4. Execute AFTER declaratives.
- Output File - LABEL RECORDS ARE *data-name*
  1. Move spaces to label record character positions 5 through 80.
  2. Execute BEFORE declarative.
  3. Move data to user label fields according to the VALUE OF clause.
  4. Execute AFTER declarative.
  5. Examine label identifier to determine if a proper label was created, and if so, write the label.

3.3.6. The WRITE Statement

Format

```
WRITE record-name [FROM identifier-1] INVALID KEY imperative-statement
```

3.3.6.1. Tape and Mass Storage Files

Execution of the WRITE statement releases a logical record to an output or an I-O file and permits performance of an *imperative-statement* if the limits specified by the FILE-LIMIT clause or the maximum address assigned to the file are exceeded.

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

The logical record released 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-1* 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. The parameter *record-name* is the name of a logical record in the File Section of the Data Division and may be qualified.

3.3.6.2. Mass Storage Files

When a sequential mass storage file is opened for I-O and the AT END condition has been reached, the file may be extended by executing subsequent WRITE statements.

When the file is opened for I-O, all WRITE statements executed following a READ execution assume the meaning of a REWRITE. When a WRITE is not preceded by a READ, the MSCS provides positioning to the next logical record, which is then overwritten. The next logical record in this case follows the last logical record accessed by the object program, which is not necessarily the last record accessed by the cycle executing the WRITE statement.

When a WRITE statement is overwriting an original record, the length of the updated logical record can be no longer than the length of the original record.

3.3.6.3. Tape Files

When a WRITE statement causes the end of the reel to be reached on a labeled tape file (except for LION, CFH. SDF. FORM01, FORM02. and FORM03), the following label processing occurs:

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. A hardware EOF mark is written to tape.
  2. An EOF1 label record is created with standard default values.
  3. The BEFORE ENDING REEL declarative is executed.
  4. The VALUE OF items are moved to the label record.
  5. The AFTER ENDING REEL declarative is executed.
  6. The EOF1 and EOF2 label records are written to tape followed by two hardware EOF marks.
  7. A tape reel swap is performed.
  8. The BEFORE BEGINNING REEL declarative is executed.
  9. The AFTER BEGINNING REEL declarative is executed.
  10. The HDR1 label record is created with standard default values.
  11. The VALUE OF items are moved to the label record.
  12. The HDR1 label is written to tape followed by the HDR2 label followed by a hardware EOF mark.

  For tape file sets using the 8-bit tape labeling system, it is not possible to change the values in the EOF1 label in step 4 or the HDR1 label in step 11. The system will use values compatible with those used when the HDR1 label was written at the time the file was opened.

- If the LABEL RECORDS ARE *data-name* clause is present:
  1. A hardware EOF mark is written to tape.
  2. An EOF1 label record is created with the standard default values.
  3. The VALUE OF items referencing the standard label names are moved to the label record.
  4. The EOF1 label record is written to tape, followed by the EOF2 label.
  5. The label record, character positions 5 through 80, is cleared to spaces.
  6. The BEFORE ENDING REEL declarative is executed.
  7. The VALUE OF items referencing user label names are moved to the label record.
  8. The AFTER ENDING REEL declarative is executed.
  9. The first three bytes of the label record are examined as follows:
     a. If they contain the characters UTL, the label is written to tape and processing is repeated starting at step 5.
     b. If the first three characters of the label area do not contain UTL, or if the first four characters remain unchanged after the reiterative process, then ending REEL label processing is complete and processing continues at step 10.
  10. Two hardware EOF marks are written to tape, and a tape reel swap is performed.
  11. The label record, character positions 5 through 80, is cleared to spaces.
  12. The BEFORE BEGINNING REEL declarative is executed.
  13. The VALUE OF items referencing user label names are moved to the label record.
  14. The AFTER BEGINNING REEL declarative is executed.
  15. The first three characters of the label record are examined as follows:
     a. If they contain the characters UVL, the label is written to tape and processing is repeated starting at step 11.
     b. If they do not contain UVL, or if the first four characters remain unchanged after the reiterative process, then beginning REEL label processing is complete, and processing continues at step 16.
  16. The HDR1 is created with standard default values.
  17. The VALUE OF items referencing standard label names are moved to the label record area.
  18. The HDR1 label is written to tape followed by the HDR2 label, followed by a hardware EOF mark.

For tape file sets using the 8-bit tape labeling system:

- Step 15 is not performed.
- It is not possible to change the values of the standard label items in the EOF1 label (step 3) or the HDR1 label (step 17). The system will use values compatible with those used when the HDR1 label was written at the time the file was opened.

Unlabeled tapes cause a hardware EOF mark to be written, and a tape reel swap is performed.

On LION files, the end-of-reel label block is written to tape followed by two hardware EOF marks. A tape reel swap is performed. A header label block (unless LABELS ARE OMITTED) is written.

On CFH files, the end-of-reel label block is written followed by two hardware EOF marks. A tape reel swap is performed. A header label block and a hardware EOF mark are written.

On FORM01 files, two hardware EOF marks are written. A tape reel swap is performed.

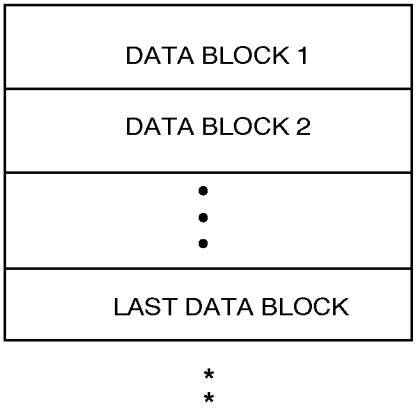
On FORM02/FORM03 files, a hardware EOF mark is written then tie end-of-reel label block followed by two hardware EOF marks. A tape reel swap is performed. A header label block is written followed by a hardware EOF mark.

On SDF files, the end-of-reel labels are processed by the Processor Common Input/Output System (PCIOS). No declaratives are executed for these files.

3.4. 1968 Standard COBOL Tape Files

3.4.1. Unlabeled File Structure

Unlabeled tape files consist only of data blocks. The last data block on a tape reel is followed by two hardware EOF marks.



Note: ** represents hardware EOF mark*

3.4.2. Labeled File Structure

American National Standard tape labels are defined in the American National Standard documents X3.2/759, X3.2.5/128 and the revision of X3.2/513 dated July 18,1968.

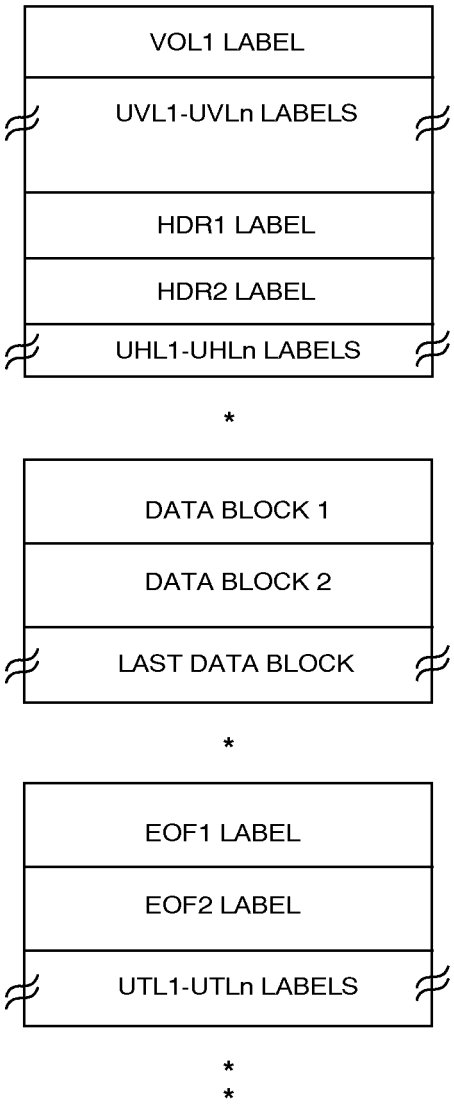
Type	Identifier and Number	Created By	Name	Required/ Optional
Beginning of Volume	VOL 1	System Tape Handler	Volume Header Label	Required
	UVL1 to UVL9 UVLA to UVLZ[3]	User	User Volume Header Label	Optional[2]
End of Volume	EOV1	System Tape Handler	End-of-Volume Label	Required
	EOV2	System Tape Handler	2nd End-of-Volume Label	Optional[1]
	UTL1 to UTL9 UTLA to UTLZ[3]	User	User End-of-Volume Label	Optional
Beginning of File	HDR1	System Tape Handler	1st File Header Label	Required
	HDR2	System Tape Handler	2nd File Header Label	Optional[1]
	UHL1 to UHL9 UHLA to UHLZ[3]	User	User File Header Label	Optional
End of File	EOF1	System Tape Handler	End-of-File Label	Required
	EOF2	System Tape Handler	2nd End-of-File Label	Optional[1]
	UTL1 to UTL9 UTLA to UTLZ[3]	User	User End-of-File Label	Optional

1 An HDR2, EOF2, and EOF2 label is always written to an output file. For 9-bit labeled tapes, these labels are optional on an input file. For 8-bit labeled tapes, these labels are required on an input file.

2 Tape file sets using the 8-bit tape labeling system do not have UVL labels.

3 Tape file sets using the 8-bit tape labeling system must have a digit (1-9) in the fourth character position of a user label identifier.

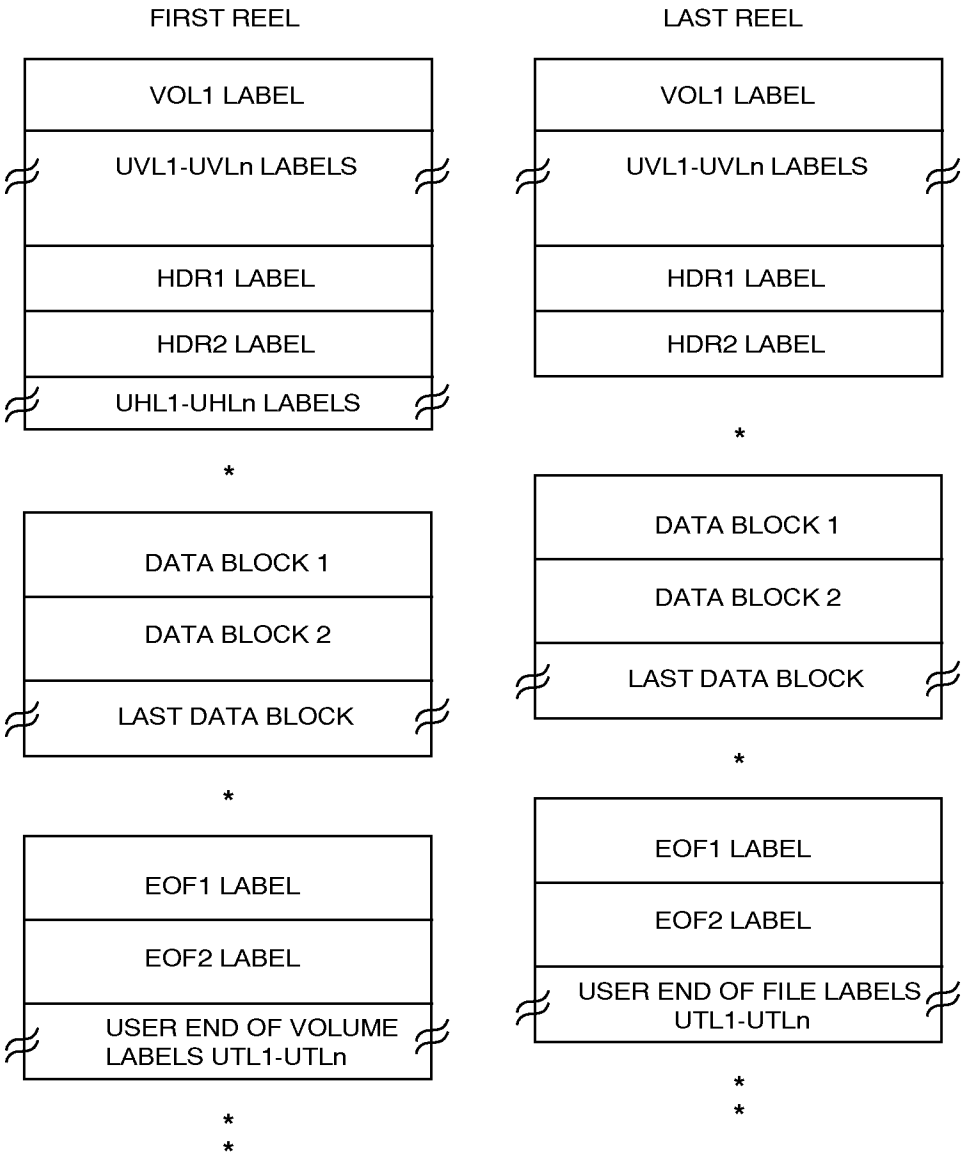
3.4.2.1. Single-Reel Single-File Structure



Note: ** represents hardware EOF mark.*

Tape file sets using the 8-bit tape labeling system do not have UVL labels.

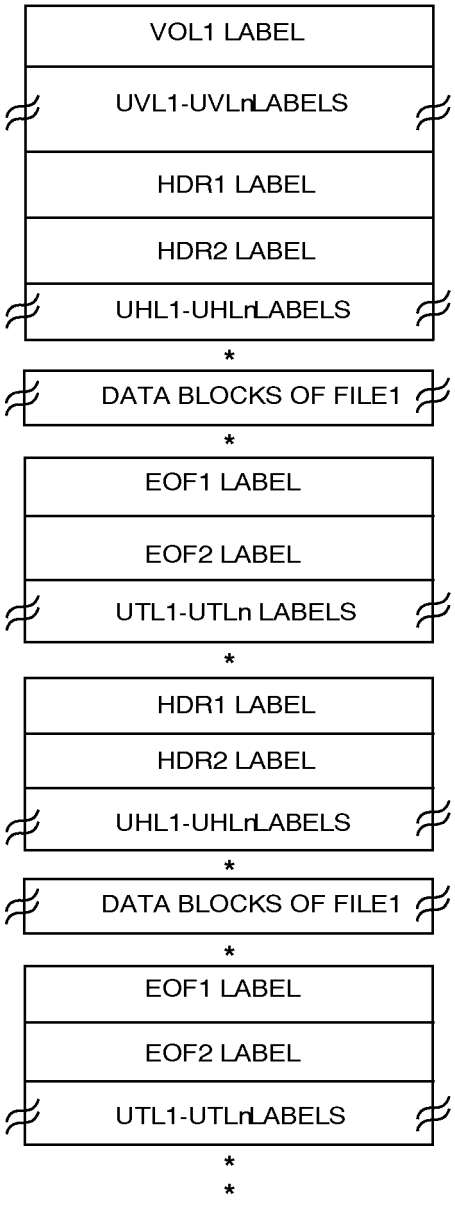
3.4.2.2. Multiple-Reel Single-File Structure



Note: ** represents hardware EOF mark.*

Tape file sets using the 8-bit tape labeling system do not have UVL labels.

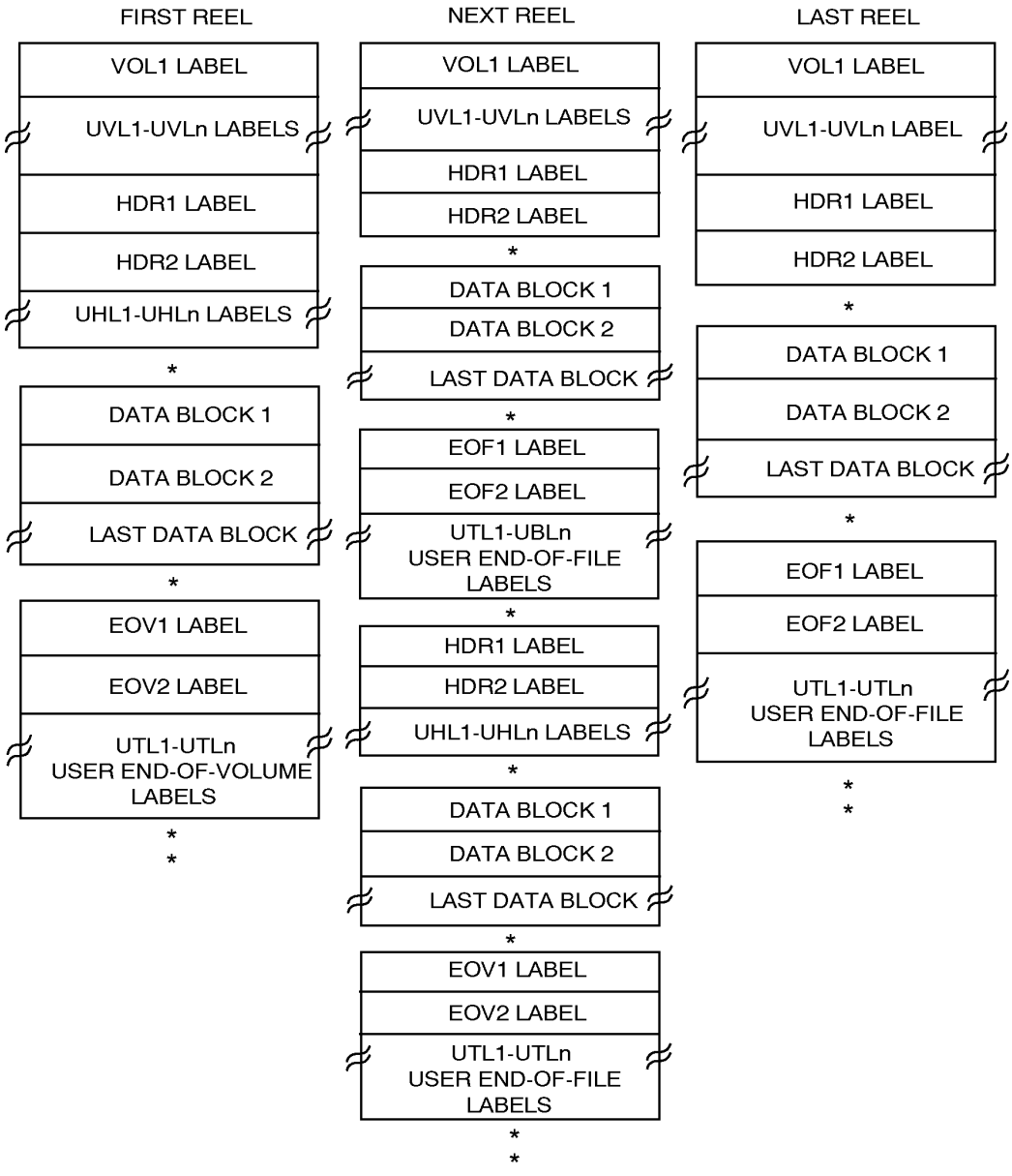
3.4.2.3. Multiple-File Single-Reel Structure



Note: ** represents hardware EOF mark.*

Tape file sets using the 8-bit tape labeling system do not have UVL labels.

3.4.2.4. Multiple-Reel Multiple-File Structure



Note: ** represents hardware EOF mark.*

Tape file sets using the 8-bit tape labeling system do not have UVL labels.

3.4.3. Label Record Formats

All labels are 80 characters in length. Labels written with the 9-bit tape labeling system are written in 9-bit ASCII. Labels written with the 8-bit tape labeling system are written in 8-bit ASCII.

3.4.3.1. Volume Header Label (VOL1)

The VOL1 label exists as the first label block on every reel. It is read by the Exec to verify that the proper reel has been mounted.

The COBOL program has no access to this label.

Character Position	Length	Standard Content	Description
1-3	3	VOL	Label identifier
4	1	1	Label number
5-10	6	Reel number	Six alphanumeric characters identifying the physical reel
11	1	Blank or nonblank	Nonblank indicates restricted access, as the tape is privately owned
12-37	26	Spaces	Not used
38-51	14	Owner Identification	Spaces
52-79	28	Spaces	Not used
80	1	1	Indicates tape contains standard labels

3.4.3.2. First File Header Label (HDR1)

Character Positions	Length	Standard Default Values	Description	VALUE OF name
1-3	3	HDR	Label identifier	None
4	1	1	Label number	None
5-21	17	External-file-name	File identifier	FILE-ID
22-27	6	Univac for 9-bit reel number from first reel is set for 8-bit.	Set identifier	SET-ID (9-bit only)
28-31	4	0001	File section number	None
32-35	4	0001	File sequence number	None
36-39	4	0001	Generation number	None
40-41	2	00	Generation version number	None
42-47	6	Date when file created	Creation date in Julian form preceded by one space	CREATION-DATE (9-bit only)
48-53	6	Creation date plus period from ASG statement	Purge date in Julian form preceded by one space	PURGE-DATE
54	1	Space	Accessibility	FILE-ACCESS
55-60	6	000000	Block count	None
61-73	13	File qualifier from ASG statement for 9-bit: “U1100-1” for 8-bit	File qualifier	FILE-QUALIFIER (9-bit only)
74-80	7	Spaces	Unused	None

As noted in the previous description, the COBOL program may set values into an HDR1 label of an output file by using the VALUE OF clause specifying the names listed in the right-hand column. For tape file sets using the 8-bit tape labeling system, the fields SET-ID, FILE-QUALIFIER, and CREATION-DATE cannot be set by the user.

3.4.3.3. Second File Header Label (HDR2)

Character Position	Length	Standard Default Values	Description
1-3	3	HDR	Must be HDR
4	1	2	Must be 2
5	1	D	Variable record format with the number of words in the record specified in decimal
6-10	5	Block length	Five digits specifying the maximum number of characters per block
11-15	5	Record length	Maximum record length including any count fields
16-50	35	Reserved for operating system	Reserved for operating systems use. Any alphanumeric characters.
51-52	2	Buffer offset (optional)	Two digits specifying the length in characters of any additional field inserted before a data block, for example, block length. This length is included in the block length (characters 6-10).
53-80	28	Spaces	Must be spaces

This label record is written on all output files following the HDR1 label. It may not be modified by a COBOL program. It is included for information interchange purposes.

3.4.3.4. End-of-File Label (EOF1)

The EOF1 label has the same format as the HDR1 label. The standard default values are the same, except for the block count field, which is set to the number of physical data blocks since the previous HDR label group.

The VALUE OF clause causes the fields to be modified on output or verified on input in a manner identical to that described for the HDR1. The Exec performs no verification of this label.

For tape file sets using the 8-bit tape labeling system, fields in the end-of-file labels cannot be modified from those of the header labels.

3.4.3.5. Second End-of-File Label (EOF2)

The EOF2 label has the same format as the HDR2 label, except for the label identification. This label is written on all output tapes following the EOF1 label.

3.4.3.6. End-of-Volume Label (EOF1)

The EOF1 label has the same format as to the EOF1 label, except for the label identification. This label is written on all output tapes which are nonfinal volumes of the file.

3.4.3.7. Second End-of-Volume Label (EOF2)

The EOF2 label has the same format as the EOF2 label except for the label identification. This label is written on all output tapes which are nonfinal volumes of the file. It is written following the EOF1 label.

3.4.3.8. User Labels

In addition to the system standard labels, Standard COBOL 1968 tape files may optionally include user labels.

In order to create user labels on an output file, the COBOL program must include the LABELS ARE *data-name* clause. User labels are then created by the use of the VALUE OF *data-name* clause, by the specification of one or more USE...LABEL PROCEDURE declarative sections, or by both.

The VALUE OF processing is performed each time a user label may be created. No distinction is made between references to *data-name* clauses in different label record descriptions. The user must ensure that the VALUE OF items are meaningful at the time a user label may be created.

A valid user label conforms to the following description:

Character Position	Length	Name	Required Contents
1-3	3	Label identifier	UVL for user beginning volume label (9-bit only) UHL for user beginning file label UTL for user end-of-volume or end-of-file label
4	1	Label number	1 through 9 (or A through Z for 9-bit only) with no two consecutive labels having positions 1-4 equal
5-80	76	As defined by user	May be set to any ASCII number or character string

The following chart indicates the time at which it may be created or checked, and the sequence of processing.

Label Identifier	Events Causing Label Processing to Occur	Sequence of Label Processing
UVL-User beginning volume labels	OPEN CLOSE REEL (after tape swap) End of reel on READ or WRITE	1. BEFORE BEGINNING REEL declarative is executed. 2. VALUE OF processing. 3. AFTER BEGINNING REEL declarative is executed.
UHL-User file header labels	OPEN	1. BEFORE BEGINNING FILE declarative is executed. 2. VALUE of processing. 3. AFTER BEGINNING FILE declarative is executed.
UTL-User end-of-volume labels	End of reel on CLOSE file CLOSE REEL on output file End of reel on READ or WRITE	1. BEFORE ENDING REEL declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING REEL declarative is executed.
UTL-User end-of-file labels	CLOSE	1. BEFORE ENDING FILE declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING FILE declarative is executed.

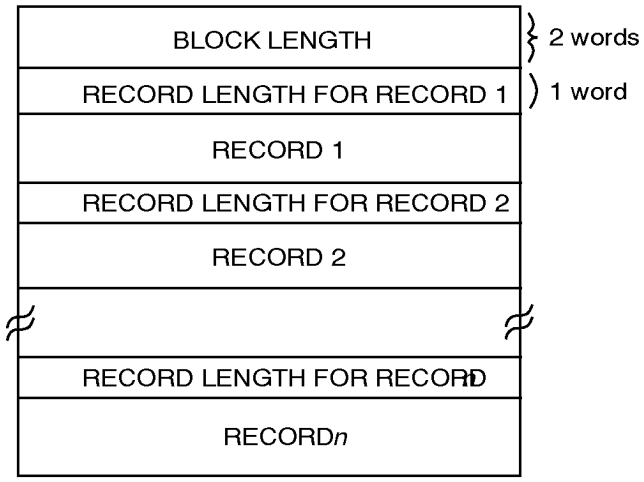
For tape file sets using the 8-bit tape labeling system, UVL labels do not exist, and the label number (character position 4) of a user label must be a digit.

For output files, the user label is written after step 3 of the sequence of label processing shown only if the label identifier is set correctly in the label record area. When a valid label exists, it is written to tape and steps 1 through 3 are repeated. Additional user labels can be created by adding 1 to the label number following the label identifier. If the label identifier and label number remain unchanged after a reiteration, no label is written and the label processing for that particular identifier is complete.

For input files, steps 1 through 3 are performed only if a label with that appropriate identifier exists on tape. Steps 1 through 3 are performed for every label that is read with the correct identifier.

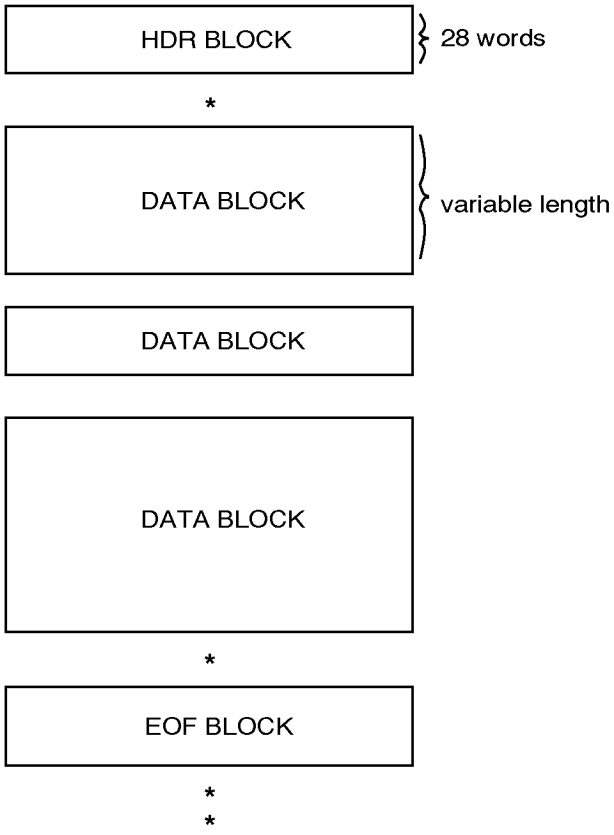
3.4.4. Data Block Format

Data blocks are variable in length, with a record length word preceding each logical record, and a 2-word block length at the beginning of a block. The length fields are recorded in 9-bit ASCII code, representing the length in decimal word. In the case of record length, the record length word is counted as part of the record length.



3.5. CFH Files

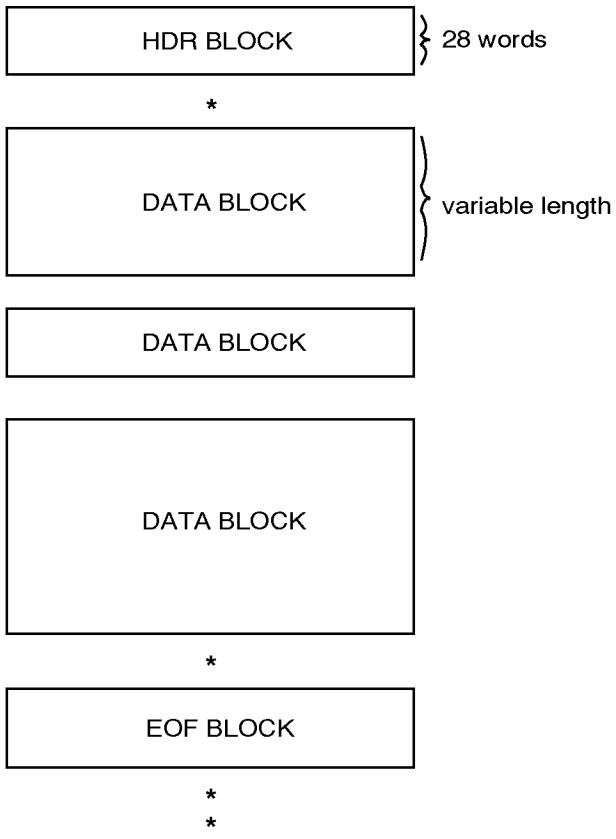
3.5.1. Unlabeled File Structure



Note: *Note: * represents hardware EOF mark.*

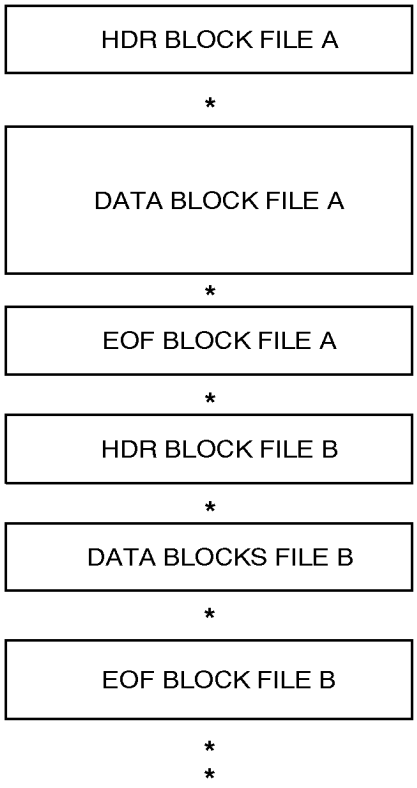
3.5.2. Labeled File Structure

3.5.2.1. Single-Reel Single-File Structure



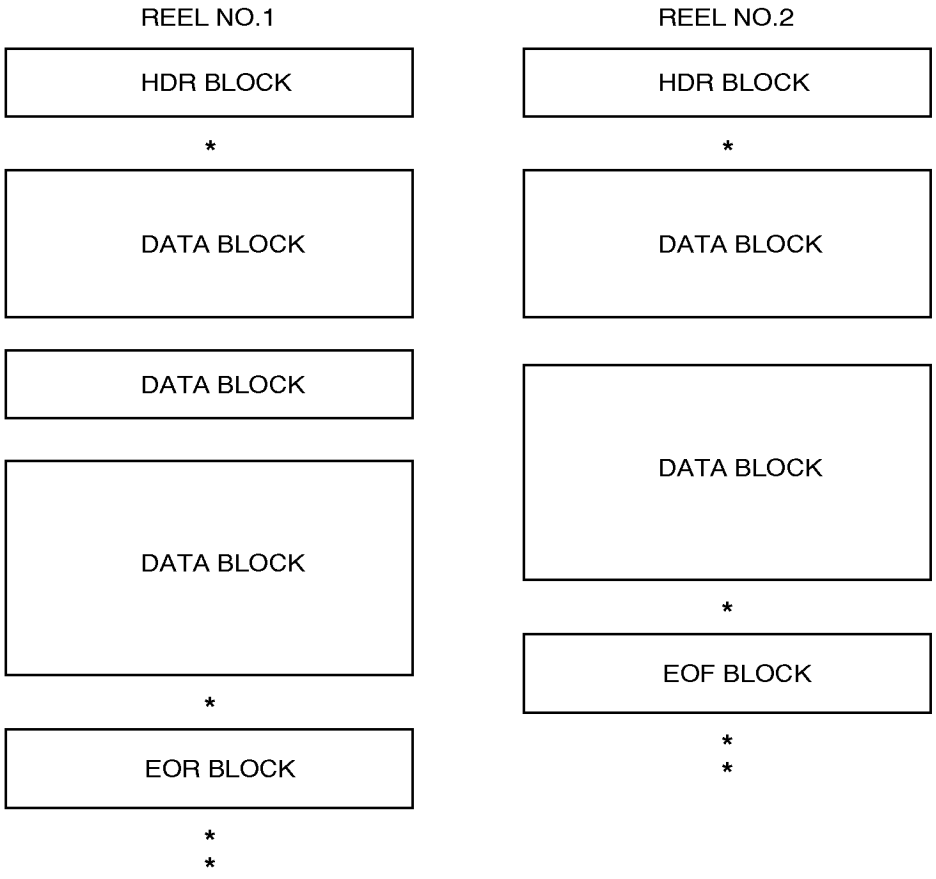
Note: ** represents hardware EOF mark.*

3.5.2.2. Single-Reel Multiple-File Structure



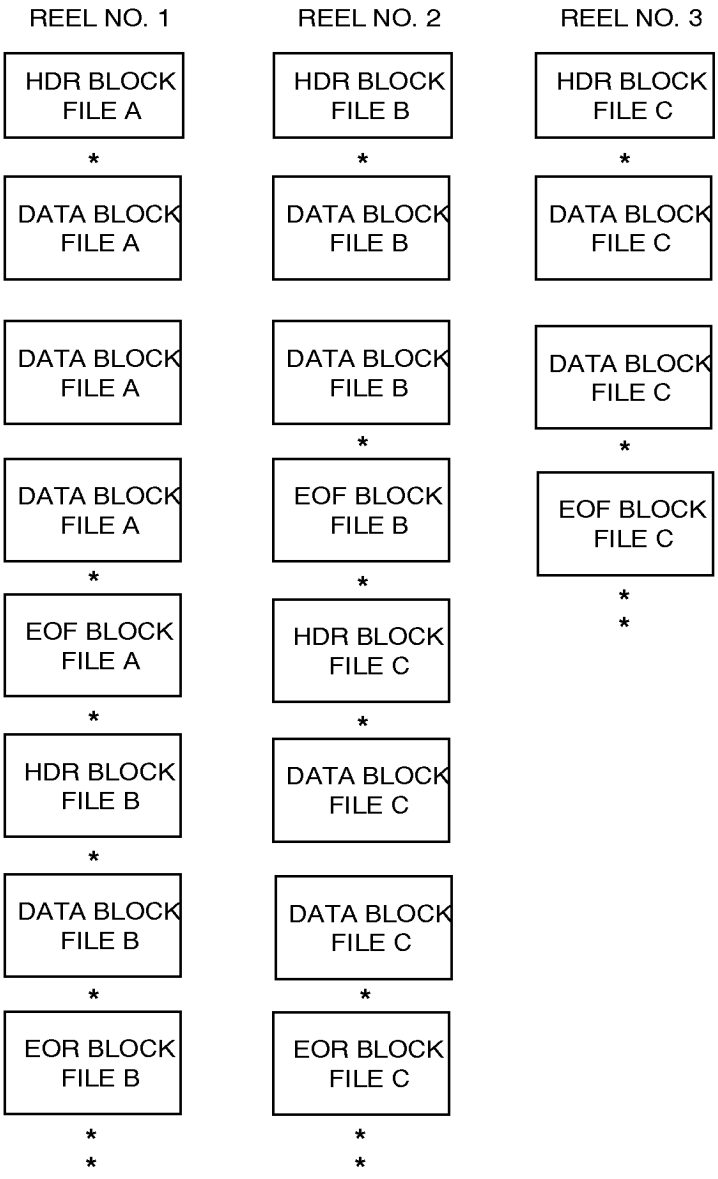
Note: ** represents hardware EOF mark.*

3.5.2.3. Multiple-Reel Single-File Structure



Note: ** represents hardware EOF mark.*

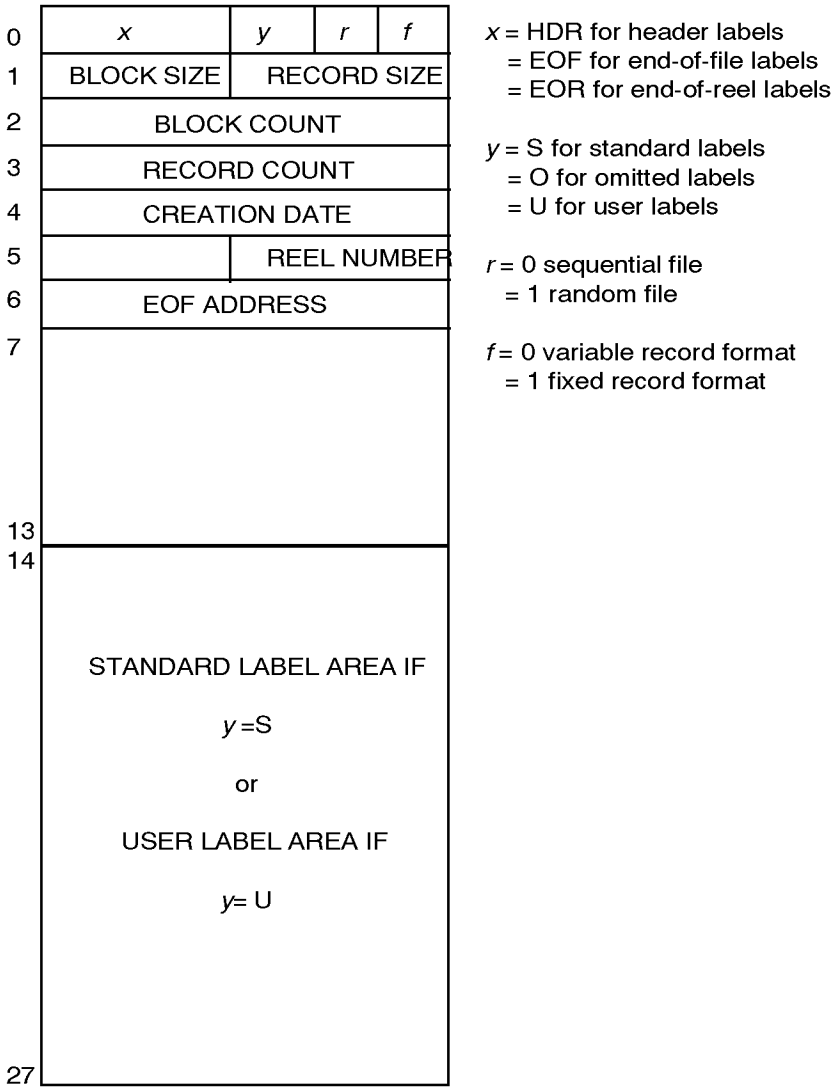
3.5.2.4. Multiple-Reel Multiple-File Structure



Note: ** represents hardware EOF mark.*

3.5.3. Label Record Formats

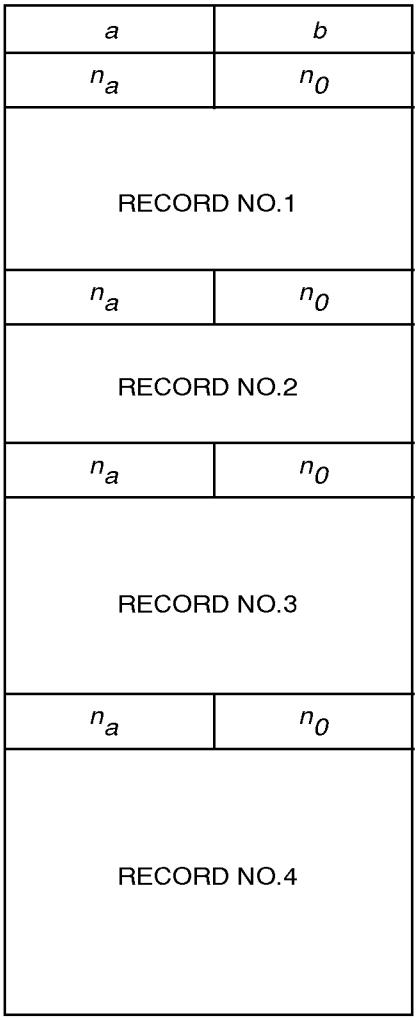
All CFH label blocks are 28 words in length and have the following format:



Note: *Only words 0, 1, 2, and 3 are recognized and processed by ASCII COBOL. For CFH files created by ASCII COBOL, field y is always set to 0. This prevents Fieldata COBOL CFH from checking more than the first four words of the label.*

3.5.4. Data Block Format

All CFH files have the same data formats. The first word of each contains a block record count and block size. In addition, each record of the block has a single-word header with the actual record size in words.



a = block record count
b = block size in words
(includes control words)
(blocks variable in length)

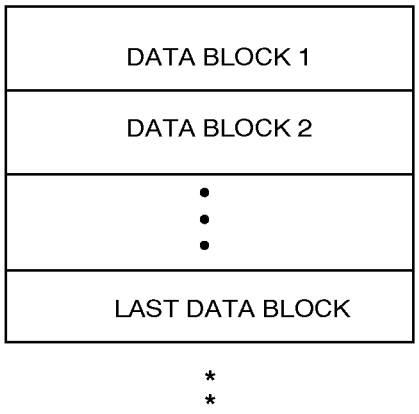
n_a = actual record size in words
n_0 = original record size in words

$n_0 = n_a$ for tape files

3.6. Compatible Files

3.6.1. Unlabeled File Structure

Unlabeled tape files consist only of data blocks. The last data block on a tape reel is followed by two hardware EOF marks.



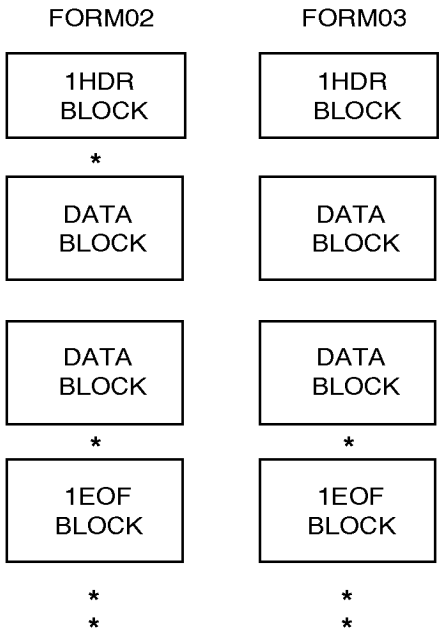
Note: ** represents hardware EOF mark.*

3.6.2. Labeled File Structure

The two types of labeled compatible file structures available are:

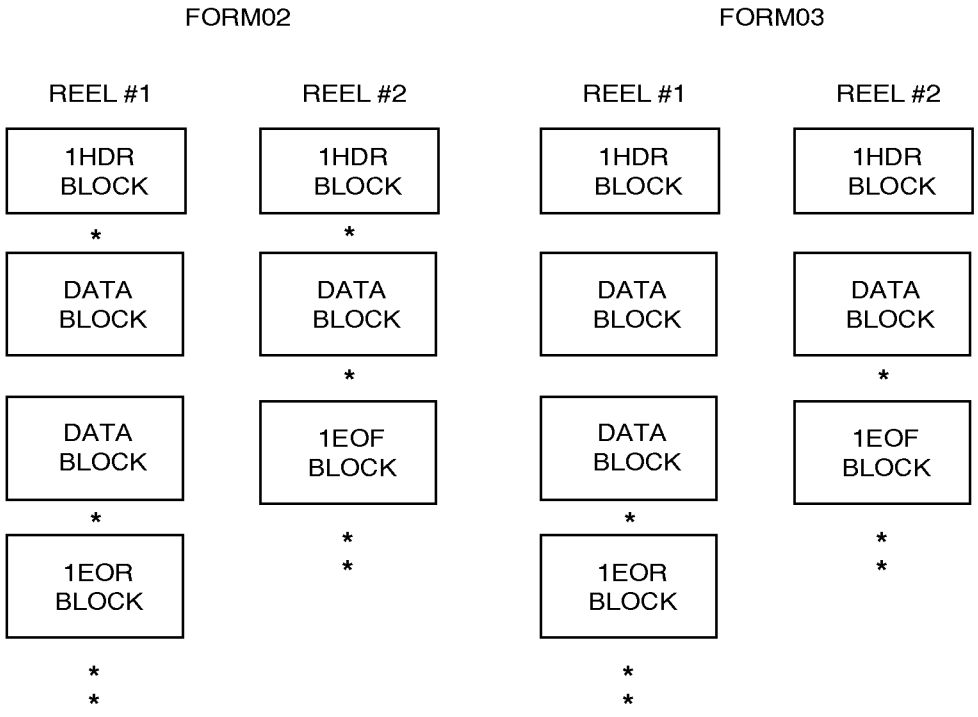
- FORM02 reads or writes files that contain label and trailer blocks with a hardware mark recorded after the label block and surrounding the trailer blocks.
- FORM03 is identical to FORM02 except that the hardware mark is omitted after the label block.

3.6.2.1. Single-Reel File Structure



Note: ** represents hardware EOF mark.*

3.6.2.2. Multiple-Reel File Structure



Note: ** represents hardware EOF mark.*

3.6.3. Label Record Formats

Label blocks are considered 80 characters in length, but 84 characters (14 words) are actually written to tape. The last four characters of word 14 are 046 characters.

When the LABEL RECORDS ARE STANDARD clause is used, the first four characters of the space-filled label are one of the following:

- 1HDR for header labels
- 1EOR for end-of-reel labels
- 1EOF for end-of-file labels

These are written on output and checked on input.

3.6.4. Data Block Format

Compatible files have two formats. One is used for fixed-length records and one is used for variable length records.

3.6.4.1. Fixed-Length Format

A file is considered to contain fixed-length records when all the records described for the file are exactly the same length (characters).

Each block contains a fixed number of records. Each record immediately follows the previous record in the block.

The last block is padded with records of all 9s when necessary to fill the block. These records are given to the user as valid records on input.

Fixed word length blocks are written. The last word of each block may contain one to five Fieldata 046 characters when the actual block size is not a multiple of 6 characters.

3.6.4.2. Variable-Length Format

A file is considered to contain variable-length records when there is more than one record described for a file, and they are not all the same length (characters).

Each variable-length record must have a Fieldata 077 character as the last character of the record. This 077 must be inserted in the record descriptions by the user. Each record immediately follows the previous record in the block. Records are placed in the block until the block cannot contain the largest record described in the Fieldata COBOL entry. The record is written in the next block, and the present block is padded with Fieldata 046 characters.

The last block is padded with records of all 9s until one does not fit; then the block is padded with 046 characters. The all 9s records which contain a 077 code as the last characters are given to the user as valid records on input.

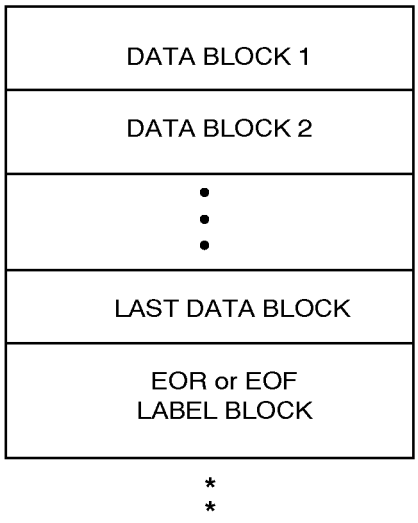
As for fixed-length records, word length blocks are written with the last word of the block padded with 046 characters if the actual block is not an multiple of 6 characters.

3.7. LION Files

A LION tape has a header label block unless the LABELS ARE OMITTED clause is specified. End-of-reel or end-of-file label blocks are always present following the last data block.

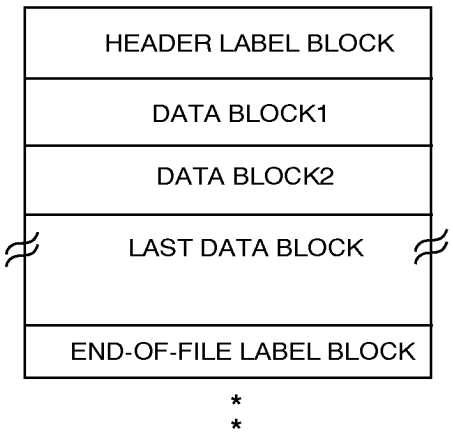
3.7.1. Unlabeled File Structure

On LION tapes, the last data block is followed by an end-of-reel or end-of-file label block, (in other words, LABEL RECORDS ARE OMITTED specified for a LION tape, applies only to the labels at the beginning of the tape). Two hardware EOF marks follow the label block.



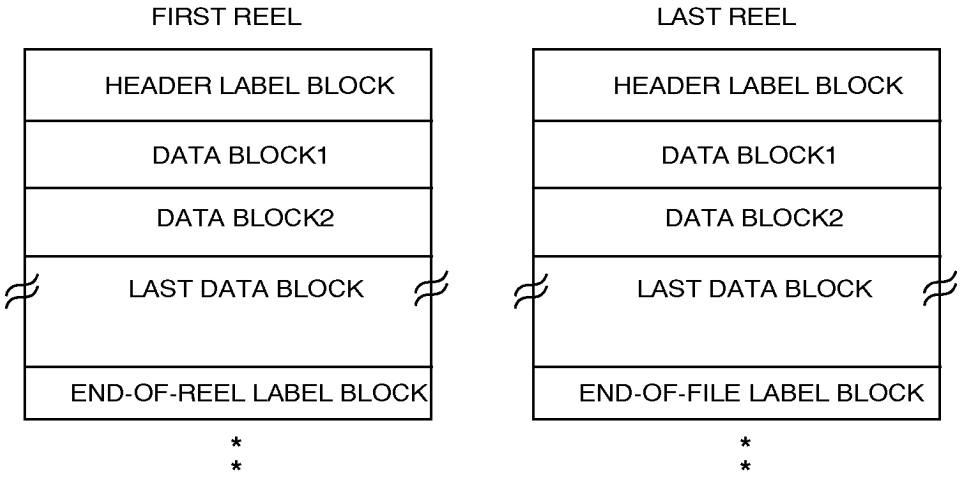
3.7.2. Labeled File Structure

3.7.2.1. Single-Reel File Structure



Note: ** represents hardware EOF mark*

3.7.2.2. Multiple-Reel File Structure



Note: ** represents hardware EOF mark.*

3.7.3. Label Record Formats

LION header label blocks are 17 words in length. End-of-reel and end-of-file label blocks are 14 words in length. A sentinel in the first word of the block identifies it as a label block. The first six bits of the second word identify the type of label.

3.7.3.1. Header Label Block

Word	Position	Description	Content
0		Sentinel	747574757475_8
1	S1	Label identifier	60_8
1	S2-S6	Block count	Zeros
2	H1	Maximum data block size	Computed from maximum record length plus BLOCK CONTAINS clause
2	H2	Fixed record length	Zero if file contains multiple record descriptions of different lengths
3	H1	Number of label words	7
3	H2	Number of free words	Zero

continued

Word	Position	Description	Content
4		Date of recording	Spaces
5	H1	Version number	4
5	H2	Reel sequence number	Reel sequence number starting with 1
6-10		Unused	Spaces
11	H1	Version number	4
11	H2	Unused	Zero
12-14		Unused	Zero
15	S1	Label identifier	60_8
15	S2-S6	Unused	Zero
16		Sentinel	747574757475_8

3.7.3.2. End-of-Reel/File Label Block

Word	Position	Description	Content
0		Sentinel	747574757475_8
1	S1	Label identifier	00 = end of file 20_8 = end of reel
1	52-56	Block count	Number of physical blocks on reel
2		Same as word 2 in header label	
3-4		Unused	Zero
5	H1	Version number	4
5	H2	Unused	Zero
6-7		Unused	Zero
8	H1	Version number	4
8	H2	Unused	Zero
9		File record count	Number of records written to file

continued

Word	Position	Description	Content
10		Checksum	Checksum for file
11-13		Same as words 2, 1, and 0	

3.7.4. Data Block Formats

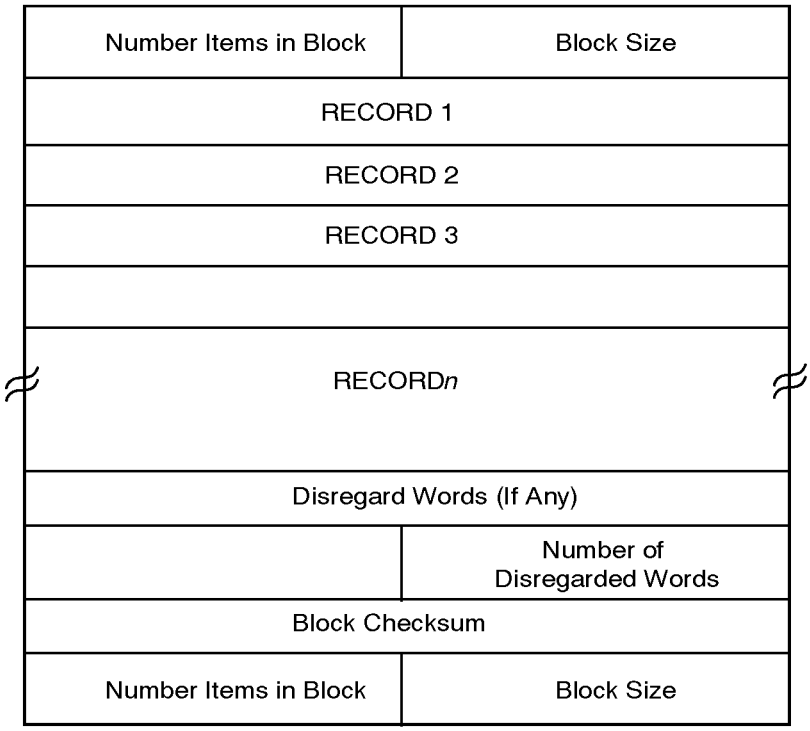
LION files have two formats. One is used for fixed-length records and one is used for variable-length records.

3.7.4.1. Fixed-Length Format

The first word of the block contains the number of logical records in the block in the left half word and the number of words in the block in the right half word.

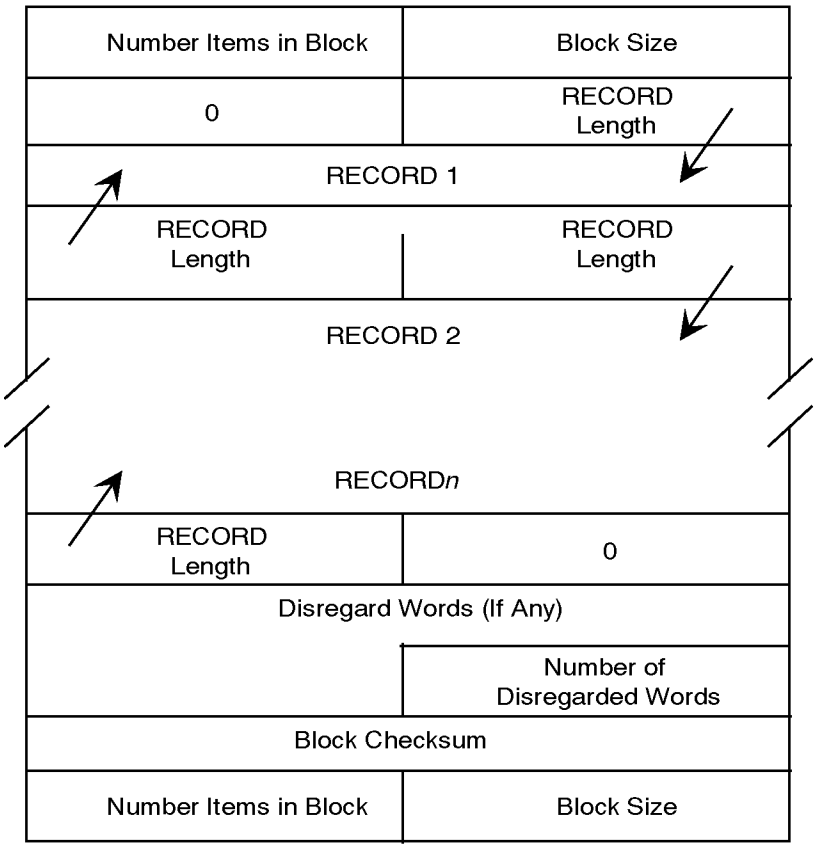
The last two words in the block contain the block checksum followed by a word repeating the first word of the block.

The minimum block size which can be written is 14 words. If a block is less than 14 words, padding words are used to pad the block to the minimum size. The right half of word 12 will contain the number of padding words.


Number Items in Block
Block Size
RECORD 1
RECORD 2
RECORD 3
RECORDn
Disregard Words (If Any)
Number of Disregarded Words
Block Checksum
Number Items in Block
Block Size

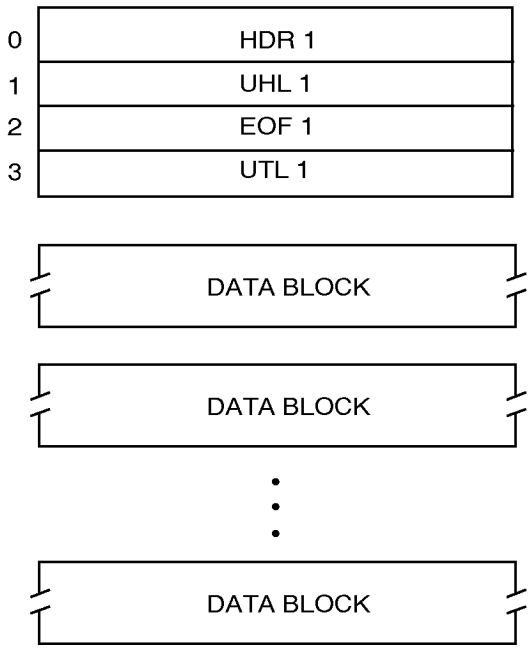
3.7.4.2. Variable-Length Format

The first word of the block contains the number of logical records in the block in the left half word and the number of words in the block in the right half word. One word is inserted preceding each logical record and following the last logical record. The left half word contains the length of the preceding record, and the right half word contains the length of the following record. The last two words of the block are identical to the fixed-length format, including the disregarded words condition.



3.8. 1968 Standard COBOL Sequential Mass Storage Files

3.8.1. Unlabeled File Structure



3.8.2. Labeled File Structure

For mass storage files, ASCII COBOL processes a subset of the tape labels which are defined in the American National Standard documents X3.2/759,X3.2.5/128, and the revision of X3.2/513, dated July 18, 1968. These labels are 28 words in length, and occupy sectors 0 through 3 of the file. The space for system labels (HDR1/EOF1) and user labels (UHL1/UTL1) is always allocated in the file, even if LABEL RECORDS ARE OMITTED is specified.

The following table gives the types of labels, their names, their identifiers, and their numbers. The table also shows which labels are required and which are permitted as options.

Type	Identifier and Number	Created By	Name	Required/ Optional
Beginning of File	HDR1	System File Handler	File Header Label	Required
	UHL1	User	User File Header Label	Optional
End of File	EOF1	System File Handler	End-of-File Label	Required
	UTL1	User	User End-of-File Label	Optional

Note: *When user label records are present, they always follow the system label of the same type.*

3.8.3. Label Record Formats

3.8.3.1. File Header Label (HDR1)

Character Positions	Length	Standard Default Values	Description	VALUE OF name
1-3	3	HDR	Label identifier	None
4	1	1	Label number	None
5-21	17	External-file-name	File identifier	FILE-ID
22-27	6	First reel number	Set identifier	SET-ID
28-31	4	0001	File section number	None
32-35	4	0001	File sequence number	None
36-39	4	0001	Generation number	None
40-41	2	00	Generation version number	None
42-47	6	Date when file was created	Creation date in Julian form preceded by one space	CREATION-DATE

continued

Character Positions	Length	Standard Default Values	Description	VALUE OF name
48-53	6	Creation date plus period from ASG statement	Purge date in Julian form preceded by one space	PURGE-DATE
54	1	Space	Accessibility	FILE-ACCESS
55-60	6	000000	Block count	None
61-73	13	File qualifier from ASG statement	File qualifier	FILE-QUALIFIER
74-80	7	Spaces	Unused	None
81-84	4	Mass-storage end-of-file address	Unused	None

3.8.3.2. End-of-File Label (EOF1)

The EOF1 label has the same format is the HDR1 label. The standard default values are the same, except for the “block count field. which is set to the number of physical data blocks contained in the file.

The EOF1 label block occupies all 28 words of sector 2 on mass storage. Words 21 through 27 (in other words, character positions 85-96) are used as needed for file control information. Current usage is as follows:

21 FASTRAND end-of-file address (sector address whole word FC$FEA)

22 Not used = zero

23 Creation time information

H1 maximum block size in words (FC$BLK)

S6 file type (FC$TYP)

24 Label flag

S6 (FC$LBL)

25 Not used = zero

26 Blocking information

H1 blocking factor (FC$BF)

S4 FC$BCC = zero if blocking factor specifies records

S4 FC$BCC = 1 if blocking factor specifies characters

27 Not used = zero

The VALUE OF clause causes the fields to be modified on output or verified on input exactly is for the HDR1 description.

3.8.3.3. User Labels

In addition to the system standard labels, mass storage files may optionally include two user labels: one UHL1 and one UTL1.

In order to create user labels on an output file, the COBOL program must include the LABELS ARE *data-name* clause. User labels are then created by the use of the VALUE OF *data-name* clause, by the specification of one or more USER LABEL PROCEDURE declarative sections, or both.

The VALUE OF processing is performed each time a user label may be created. No distinction is made between references to *data-name* clauses in different label record descriptions. It is the user's responsibility to ensure that the VALUE OF items are meaningful at the time a user label may be created.

A valid user label conforms to the following description:

Character Position	Length	Name	Required Contents
1-3	3	Label identifier	UHL for user beginning file label. UTL for user end-of-file label
4	1	Label number	1
5-80	76	As defined by user	May be set to any ASCII number or character string

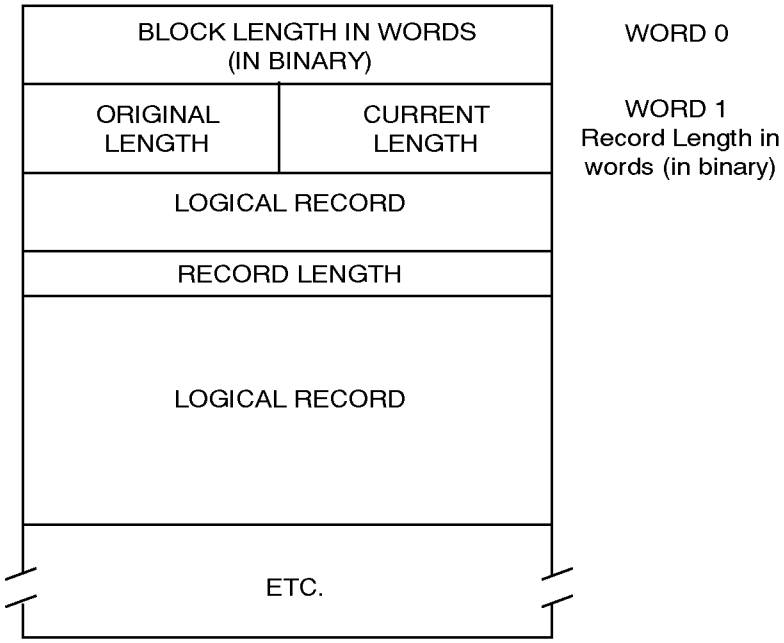
The following chart indicates the time at which it may be created or checked, and the sequence of processing.

Label Identifier	Events Causing Label Processing to Occur	Sequence of Label Processing
UHL-User file header labels	OPEN	1. BEFORE BEGINNING FILE declarative is executed. 2. VALUE of processing. 3. AFTER BEGINNING FILE declarative is executed.
UTL-User end-of-file labels	CLOSE	1. BEFORE ENDING FILE declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING FILE declarative is executed.

For output and input/output files, the user label is written only if the label identifier is set correctly in the label record area. When a valid label exists, it is written.

For input and input/output files, label processing is performed only if a label with that appropriate identifier exists.

3.8.4. Data Block Format



The data blocks start at sector 4 on all mass storage files. The first word of the block contains a binary number showing the total length of the block in words.

The second word of the block and a word preceding each logical record contain a binary number showing the original length (in words) of the logical record in H1 of this word and the current length (in words) of the logical record in H2.

3.9. SDF Mass Storage and Tape Files

For a description of the file structure, label record format, and data block format for RECORDING MODE IS SDF files, see the OS 2200 Processor Common Input/Output System (PCIOS) Administration and Programming Reference Manual, 7831 0588 (current version).

Device-Independent Files

The objective of a device-independent file is to give the user the flexibility of choosing the device on which the file resides (MASS-STORAGE or UNISERVO) by changing the @ASG statement in the run stream but without changing the source program, recompiling, and recollecting. Because of the somewhat opposing hardware characteristics of these two devices, there is a need to impose some limitations on the ORGANIZATION of files and their attributes. It is obvious that various file access techniques cannot be provided on both device types. Therefore, the ONLY type of file that can be accepted by both mass storage and tape is an ORGANIZATION IS SEQUENTIAL file.

Device independence is a characteristic of a file and an extension of the I-O facility in the ASCII COBOL language. Therefore, it is presented here as a file type with its own attributes and not as a subset of either tape or sequential mass storage.

It does not, however, provide for the total transportability of a sequential file from tape to mass storage or vice versa through some readily available utility such as the FURPUR C COPY statement.

When a file has an ASSIGN TO SEQUENTIAL-FILE clause in the FILE-CONTROL paragraph, it becomes a device-independent file. The type of I-O to perform becomes a run-time decision.

Section 4
Direct I-O

4.1. General

A file with Direct Organization has all of the attributes of a mass storage file with Sequential Organization, with the additional ability to access the file randomly.

Random access is possible on a Direct file, because each logical record is assigned mass storage equal to the size of the largest record in the file.

The ACTUAL KEY for a Direct file represents the relative number of the logical record within the file. The Mass Storage Control System (MSCS), (see the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582, (current version)) can determine the address of a specific logical record by multiplying the record size by the contents of the ACTUAL KEY. Therefore, it is not necessary to read or write the file by sequential progression from the beginning to the end.

It is not necessary, however, to always access a Direct file randomly. The MSCS allows a Direct file to be accessed sequentially. A Direct file, when sequentially accessed, is treated by the MSCS as a mass storage file with sequential organization; however, because the logical records are not grouped into physical blocks on mass storage, each READ, WRITE, etc. causes an I-O access. An advantage in organizing a file on mass storage as Direct is the use of the START statement, which allows the COBOL program to instruct the MSCS to position the file backward or forward.

4.2. Environment Division

4.2.1. The FILE-CONTROL Paragraph

4.2.1.1. The ACTUAL KEY Clause

Format

```
[ ACTUAL KEY IS  data-name-1]
```

Description

When ACCESS IS SEQUENTIAL, this clause is only required if the START statement is to be executed for the file. If the ACTUAL KEY clause is specified, the contents are updated by the MSCS following each READ statement execution.

When ACCESS IS RANDOM, the ACTUAL KEY clause is required.

The parameter *data-name-1* may be defined as any numeric data item. The contents of *data-name-1* are moved and converted to a separate 9(10) COMP item for use by the MSCS. The contents of *data-name-1* must not cause an overflow when the conversion is performed.

The actual key is a logical record number which represents the position of the logical record within the file.

4.2.1.2. The ASSIGN Clause

Format

```
ASSIGN TO  implementor-name-1[,implementor-name-2]...
```

Description

Implementor-name-1 may be one of the following:

MASS-STORAGE
MASS-STORAGE-28
MASS-STORAGE-56
MASS-STORAGE-112

In the form MASS-STORAGE-*nnn*, the integer value *nnn* refers to the prepping factor of the mass storage file. These alternatives are available to allow blocking of records to be sensitive to different physical record sizes on disk devices. The block size will be calculated according to the BLOCK CONTAINS clause and then rounded up to the next multiple of the specified physical record size. With the *implementor-name* MASS-STORAGE, a default value of 28 words is used in rounding up block size (unless the user changes FC$PREP to 56 or 112, then 56 or 112 is used).

If the parameter *implementor-name*-2 is specified, it is a 1- to 12-character name used to link the COBOL file to Exec file control and should appear in an @ASG or @USE statement in the run stream for executing the object program.

When the parameter *implementor-name*-2 is not specified, the first 12 characters of the file-name are used to link the COBOL file to the Exec file control.

All recursions of *implementor-name-1*, *implementor-name*-2 are for documentation only.

4.2.1.3. The FILE-LIMITS Clause

Format

```
[ ,{FILE-LIMIT IS   } { data-name-1 } { THRU    } { data-name-2 }
   {FILE-LIMITS ARE} { literal-1   } { THROUGH } { literal-2   }

    [ , { data-name-3 } { THRU    } { data-name-4 } ] ... ]
        { literal-3   } { THROUGH } { literal-4   }
```

Description

The FILE-LIMITS clause specifies that logical records are retrieved or placed in the file within these limits. Each pair of operands associated with the keyword THRU represents a logical segment of the file. The logical beginning of the file is that address represented by the first operand of the first or only pair of the FILE-LIMITS clause. The logical end of the file is that address represented by the last operand of the pair of the FILE-LIMITS clause.

The parameters *data-name-1*, *literal-1*, etc., represent the relative number of a record within the file.

When a file is sequentially accessed, the MSCS obtains or places each logical record by progression from one file limit segment to another.

When a file is randomly accessed, the INVALID KEY clause is executed on any statement when the contents of the ACTUAL KEY are not within the file limits.

The FILE-LIMITS clause may not be specified for RECORDING MODE SDF files.

4.2.1.4. The PROCESSING MODE Clause

Function

The PROCESSING MODE clause specifies the order in which logical records are processed by the COBOL program.

Format

```
[ , PROCESSING MODE IS SEQUENTIAL ]
```

Description

The PROCESSING MODE clause specifies the order in which logical records are processed by the COBOL program.

When PROCESSING MODE IS SEQUENTIAL is specified, the logical records are normally processed in the order in which they are accessed. In this case, only one area is allocated to contain a logical record. If the COBOL program has reference to such a file in a USE FOR RANDOM PROCESSING declarative section, appropriate lock techniques must be used by the programmer to ensure the integrity of the logical record

if simultaneous accesses to it are possible. If this clause is absent, SEQUENTIAL PROCESSING is assumed.

4.2.1.5. The RESERVE Clause

Format

```
[ , RESERVE { integer-2 } ALTERNATE [ AREA  ] ]
            { NO        }          [ AREAS ]
```

Description

The RESERVE clause is used to specify the number of buffer storage areas to be used in processing the file.

For sequentially accessed and synchronously processed files; in other words, those for which PROCESSING MODE IS SEQUENTIAL (see 4.2.1.4), a maximum of two buffer areas is allocated, regardless of the number specified by *integer-2*. For sequential INPUT-OUTPUT (I-O) files that are being randomly processed (in other words, contain the PROCESSING MODE IS RANDOM clause; see the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582, (current version)), *integer-2* buffer areas are allocated. If the RESERVE clause is not specified, two buffer areas are allocated. If NO is specified, only one buffer area is allocated.

When ACCESS MODE IS RANDOM, the RESERVE clause is ignored since records are read and written directly into arid out of the logical record area.

Buffer areas are not compiled as part of the object program. Instead, during execution, the object program is dynamically extended to allow for the buffer areas when the file is opened. The buffer areas are released when the file is closed.

The RESERVE clause may not be specified for RECORDING MODE IS SDF files.

4.2.2. The I-O-CONTROL Paragraph

Function

The I-O-CONTROL paragraph provides for the protection of record areas.

Format

```
I-O-CONTROL.

   [; APPLY INTERLOCK ON file-name-1[,file-name-2]...]

   [; APPLY PROTECT ON file-name-3[FOR integer-1RECORDS ]

      [,file-name-4 [FOR  integer-2 RECORDS ]]...]
```

Syntax Rules

1. The I-O-CONTROL paragraph is optional.

2. The APPLY INTERLOCK clause and the APPLY PROTECT clause must not specify the same *file-name*.
3. The values of *integer-1*, *integer-2*... must be less than or equal to the value of the integer specified in the PROCESSING MODE clause. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582, (current version).

General Rules

1. The APPLY INTERLOCK clause and the APPLY PROTECT clause may be specified only for files which meet the following criteria:
   a. Opened in the I-O mode.
   b. Direct file in random or dynamic access mode.
   c. The File-Control paragraph must contain the PROCESSING IS RANDOM clause.
   d. The File Description does not include the RECORDING MODE IS SDF clause.
2. Whenever a keyed record is read from a file with the APPLY INTERLOCK clause, that record is "locked out" until:
   a. A record is read from that file;
   b. A record is rewritten to that file;
   c. The record is freed; or
   d. The controlling random processing cycle is complete.

   Any other program or cycle within the same COBOL run unit attempting to read a "locked out" record will be suspended until the lock is removed.
3. For a file with the APPLY INTERLOCK clause, the number of records concurrently locked for the file is equal to the number of concurrently active cycles accessing the file.
4. The values *integer-1*, *integer-2*... of the APPLY PROTECT clause specify the maximum number of records to be locked out concurrently. The number of concurrently locked records may be less than the number of active cycles accessing the file.

4.3. Data Division

4.3.1. The BLOCK CONTAINS Clause

Format

```
[;BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS    }]
                                            { CHARACTERS }
```

Description

For RECORDING MODE IS SDF files, this clause is ignored for ORGANIZATION DIRECT. The physical block size is calculated as follows:

```
BS = (MRL/2047)+MRL+1+222
```

where:

MRL Maximum Record Length

BS Block Size rounded up to the next higher integer.

4.3.2. The LABEL RECORDS Clause

Format

```
; LABEL { RECORD IS  } { STANDARD                                    }
       { RECORDS ARE } { OMITTED                                     }
                       { data-name-1 [ ,data-name-2 ] ... }
```

Description

Label records are located on the first four sectors of mass storage allocated to the file. This clause permits the identification of these label records. The LABEL RECORDS clause must appear for each file description.

If the user desires only the system standard label records, then the STANDARD phrase may be used. The OMITTED phrase specifies that the COBOL program does not require labels for the file. The file handler will generate labels for an output file with system default values.

The parameter *data-name-1* is the name of a label record which must be the subject of a Record Description entry associated with the file. The *data-name* must not appear in the DATA RECORDS clause of the File Description. When user label records are specified (*data-name-1*, *data-name-2* ...), the user may define fields in the description of *data-name-1* (up to 80 characters) and have values placed in them (OUTPUT) and checked for (INPUT and I-O) by using the VALUE clause. All fields described in label records must have display usage. Such fields may not be referred to outside of the USE for LABEL declarative sections in the Procedure Division.

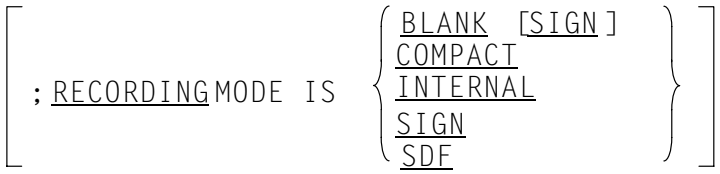
The specification of user labels indicates the presence of these labels in addition to the system standard labels.

The RECORDING MODE IS SDF files have their own labeling conventions which are independent of conventional COBOL labels. Hence, the LABEL RECORDS clause is ignored for these files.

For files without the RECORDING MODE IS SDF clause, the actual block size will be determined by the size of the largest record plus control information rounded to the next higher multiple of the prep factor specification.

4.3.3. The RECORDING MODE Clause

Format



Description

The RECORDING MODE clause specifies, for mass storage files, the format of the logical records on mass storage, which may be different from the format in main storage. Any appropriate data conversions are performed as data records are transferred between main storage and mass storage.

When RECORDING MODE IS INTERNAL or SDF is specified, it means that all records are read or written exactly as they appear in main storage with no conversion taking place. If a RECORDING MODE clause is not present, INTERNAL is assumed.

If a RECORDING MODE other than INTERNAL or SDF is specified and the file contains multiple data record descriptions. then a record selector field within each record description is required. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582, (current version) for a description of record selector fields.

When the record is read or written, the selector is tested to determine which record conversion must be performed.

4.3.3.1. The RECORDING MODE IS BLANK Clause

When the RECORDING MODE IS BLANK clause is stated, leading blanks in numeric DISPLAY or DISPLAY-1 fields are converted to leading zeroes as the records are read from mass storage.

4.3.3.2. The RECORDING MODE IS BLANK SIGN Clause

The RECORDING MODE IS BLANK SIGN clause means that the conversions for both RECORDING MODE IS BLANK and RECORDING MODE IS SIGN are to be performed on the file.

4.3.3.3. The RECORDING MODE IS COMPACT Clause

The RECORDING MODE IS COMPACT clause means that the records on the file contain variable length arrays. Each record on mass storage is compressed and is automatically converted to or from expanded form in main storage. The DEPENDING option of the OCCURS clause is used whenever a variable number of occurrences of a dimensioned item is desired. If a DEPENDING option is used within a record description of a file

whose RECORDING MODE IS COMPACT, the value of a data item is used to specify the conversion required such that unused occurrences do not appear in the external media.

If a RECORDING MODE IS COMPACT file specifies the APPLY EXDEF clause and the file contains a forward referenced OCCURS DEPENDING ON data item, the APPLY EXDEF clause can only be used in the main program.

4.3.3.4. The RECORDING MODE IS SDF Clause

The RECORDING MODE IS SDF clause specifies that the file is in the System Data Format (SDF) on the external media. The files are processed by the Processor Common Input/Output System (PCIOS).

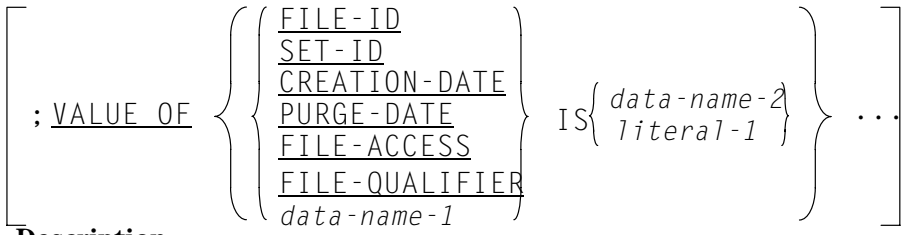
For Direct Organization files assigned to MASS-STORAGE, this file format is identical to the file format used for Relative Organization files assigned to DISC. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582, (current version).

4.3.3.5. The RECORDING MODE IS SIGN Clause

The RECORDING MODE IS SIGN clause indicates that every signed numeric DISPLAY or DISPLAY-1 data item is given an overpunched sign on the low order digit as it is read into main storage. On mass storage, the sign is carried as on explicit character preceding the high order digit of the field.

4.3.4. The VALUE OF Clause

Format



Description

The VALUE OF clause specifies the value of an item in a label record. This clause must not be present for a file when LABEL RECORDS ARE OMITTED is specified or for RECORDING MODE IS SDF files.

The following names are names of fields in the system standard labels and have the implied definition of:

	Field Name	Implied Definition
02	FILE-ID	PIC X(17)
02	SET-ID	PIC X(6)
02	CREATION-DATE	PIC 9(6)
02	PURGE-DATE	PIC 9(6)
02	FILE-ACCESS	PIC X(1)
02	FILE-QUALIFIER	PIC X(12)

In the absence of any or all of these names in the VALUE OF clause when a file is opened for output, the following values will be assumed:

FILE-ID

will be set to the 12-character external file name.

SET-ID

will be set to UNIVAC.

CREATION-DATE

will be set to the current date.

PURGE-DATE

will be set to the current date plus the expiration period specified on the @ASG statement or to operating system default if no expiration period is specified.

FILE-ACCESS

will be set to a space indicating unlimited access to the file.

FILE-QUALIFIER

will be set to the file qualifier.

In addition to the preceding system standard label names, the VALUE OF *data-name-1* IS *data-name-2* form may be used provided LABEL RECORDS ARE *data-name* is also specified. In this case, *data-name-1*, etc. must be defined in the record description for a label record.

The contents of *data-name-2* or *literal-1* will be moved to the label field on output or checked against the label field on input; *data-name-2* must be described as USAGE DISPLAY (ASCII).

When user label records are specified, the VALUE OF processing takes place in two steps:

1. When the system standard label records are being read or written, the VALUE OF standard label names are checked or moved, respectively.
2. When user label records are being read or written, the VALUE OF *data-name* clauses are checked or moved, respectively.

4.3.4.1. The VALUE OF CREATION-DATE Clause

The contents of the CREATION-DATE field are set to the current date at the time the file is opened for output. The VALUE OF CREATION-DATE clause modifies the contents of the field to the value specified. The value field is written in the COBOL program in the form *MMDDYY* and is converted to Julian form when the HDR1 label is created. At this time, the contents of the month, day, and year values are verified for a legal range of numbers.

4.3.4.2. The VALUE OF *data-name* Clause

In the VALUE OF *data-name* addition to the system standard labels, mass storage files may optionally include two user labels: one UHL1 and one UTL1.

In order to create user labels on an output file, the COBOL program must include the LABELS ARE *data-name* clause. User labels are then created by the use of the VALUE OF *data-name* clause, by the specification of one or more USER LABEL PROCEDURE declarative sections, or by both.

The VALUE OF processing is performed each time a user label may be created. No distinction is made between references to *data-name* clauses in different label record descriptions. It is the user's responsibility to ensure that the VALUE OF items are meaningful at the time a user label may be created.

The following table indicates the time at which a user label may be created (or processed on input) and the sequence of processing.

Label Identifier	Events Causing Label Processing to Occur	Sequence of Label Processing
UHL-User file header labels	OPEN	1. BEFORE BEGINNING FILE declarative is executed. 2. VALUE of processing. 3. AFTER BEGINNING FILE declarative is executed.
UTL-User end-of-file labels	CLOSE	1. BEFORE ENDING FILE declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING FILE declarative is executed.

For output files, the user label is written only if the label identifier is set correctly in the label record area. When a valid label exists, it is written.

For input files, label processing is performed only if a label with the appropriate identifier exists.

4.3.4.3. The VALUE OF FILE-ACCESS Clause

One space is moved to the FILE-ACCESS field when the HDR1 is created. The VALUE OF FILE-ACCESS clause may modify the contents of the field. A space represents unlimited access to the file.

4.3.4.4. The VALUE OF FILE-ID Clause

If specified by the VALUE OF FILE-ID clause, the file identifier is checked by the MSCS on files opened for input or I-O and moved by the file handler on output files.

When creating a file, the external file name is moved to the skeleton label. The VALUE OF FILE-ID clause may be used to modify that file name.

4.3.4.5. The VALUE OF FILE-QUALIFIER Clause

The contents of the FILE-QUALIFIER field are set originally to the qualifier specified in the @ASG statement, or the qualifier specified in the currently active @QUAL statement, or, by default, to the project field in the @RUN statement. The VALUE OF FILE-QUALIFIER clause modifies the contents of the field to the value specified.

4.3.4.6. The VALUE OF PURGE-DATE Clause

The contents of the PURGE-DATE field are set to the current date at the time the file is opened for output plus the number of days to expire specified in the @ASG statement. The VALUE OF PURGE-DATE clause modifies the contents of the field to the value specified. The rules for the value for PURGE-DATE ore the same as for CREATION-DATE.

4.3.4.7. The VALUE OF SET-ID Clause

The COBOL program may set the contents of the SET-ID field without restriction.

4.4. Procedure Division

4.4.1. The CLOSE Statement

Format

```
CLOSE file-name-1 [WITH LOCK]

  [ ,file-name-2 [WITH LOCK ]]...
```

Description

The CLOSE statement terminates the processing of one or more input or one or more output files and provides optional locking. Each *file-name* refers to an FD description in the Data Division. An OPEN statement must be executed prior to the CLOSE statement.

The CLOSE statement applies to the entire file rather than to individual units, initiates the final closing conventions for the file, and releases the data area. A file may be closed once, but not more than once, for each time the file is opened.

For an output file, the final closing conventions such as block control, etc., for the file are performed.

If the LOCK phrase is specified, the file can never be reopened in that run unit.

If the file has been specified as OPTIONAL and is not present, the standard end-of-file processing is bypassed.

A CLOSE *file-name-1* should be executed for each file that was opened.

For labeled mass storage files (except SDF), a CLOSE of the file causes the following label processing to occur:

- Input or I-O file, LABEL RECORDS ARE STANDARD
  1. The EOF1 label is read.
  2. The BEFORE ENDING FILE declarative is executed.
  3. The VALUE OF items are verified against the EOF1 label.
  4. The AFTER ENDING FILE declarative is executed.
  5. For an I-O file, the EOF1 label record is rewritten.
- Input or I-O file, LABEL RECORDS ARE *data-name*
  1. The EOF1 label record is read and verified against the VALUE OF items referencing standard label names.
  2. The EOF1 label record is rewritten.
  3. If a UTL label is present, it is read from the mass storage file and the following occurs:
     a. The BEFORE ENDING FILE declarative is executed.

b. The VALUE OF items referencing user label names are verified.

c. The AFTER ENDING FILE declarative is executed.

- Output file, LABEL RECORDS ARE STANDARD
  1. An EOF1 label record is created with standard default values.
  2. The BEFORE ENDING FILE declarative is executed.
  3. The VALUE OF items are moved to the label record.
  4. The AFTER ENDING FILE declarative is executed.
  5. The EOF1 label record is written.
- Output file, LABEL RECORDS ARE *data-name*
  1. An EOF1 label record is created with the standard default values.
  2. The VALUE OF items referencing standard label names are moved to the label record.
  3. The EOF1 label record is written.
  4. The label record, character positions 5 through 80, is cleared to spaces.
  5. The BEFORE ENDING FILE declarative is executed.
  6. The VALUE OF items referencing user label names are moved to the label record.
  7. The AFTER ENDING FILE declarative is executed.
  8. The first three characters of the label record are examined as follows:
     a. If they contain the characters UTL, the label is written.
     b. If they do not contain UTL, the label record is not written.

For RECORDING MODE IS SDF files, the end-of-file labels are processed by the Processor Common Input/Output System (PCIOS). No declaratives are executed for these files.

4.4.2. The FREE Statement

Function

The FREE statement releases a record for access by other activities.

Format

```
FREE file-name RECORD
```

Syntax Rule

The FREE statement may be specified only for files which were named in the APPLY INTERLOCK or APPLY PROTECT clause in the I-O-CONTROL paragraph (see 4.2.2).

General Rules

1. The associated file must be open in the I-O mode at the time of the execution of this statement (see 4.4.3).
2. When a record is read with protection, any other program or cycle attempting to read that record is suspended until the protection lock is removed. Normally, the protection is removed by rewriting the record or reading another record from the file. The FREE statement is used to remove the lock prior to another access to the file, so that any other cycle waiting to read that record can be resumed.
3. Following the execution of the last statement in a random processing declarative section, any records which have been read with protection are released by the file handler.
4. The value of the key associated with a direct file is not used for the FREE statement.
5. The logical record continues to be available to the COBOL program following the execution of a FREE statement.

4.4.3. The OPEN Statement

Format

```
OPEN    { INPUT  file-name-1 [ ,file-name-2]...  }
        { OUTPUT file-name-3 [ ,file-name-4]...  }  ...
        { I-O    file-name-5 [ ,file-name-6]...  }
```

Description

The OPEN statement initiates processing of named files by checking or writing labels and by performing any other input/output operations necessary prior to accessing the first record in a given file. However, the OPEN statement does not obtain or release the first data record; a READ or WRITE statement must be executed.

If an input file is designated as OPTIONAL in the FILE-CONTROL paragraph, the MSCS interrogates for the presence of the file. If the file is not present, the first READ statement for the file causes the imperative statement in the AT END phrase to be executed.

For Direct files (except SDF files or files with the LABEL RECORDS ARE OMITTED clause), the following label processing occurs:

Open Output

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. The HDR1 label is created with standard values.
  2. The BEFORE BEGINNING FILE declarative is executed.
  3. The VALUE OF processing is performed.
  4. The AFTER BEGINNING FILE declarative is executed.
  5. The HDR1 label record is written.

- If the LABEL RECORDS ARE *data-name* clause is present:
  1. The HDR1 label is created with standard values.
  2. The VALUE OF items referencing the standard label are moved to the label area.
  3. The HDR1 label is written.
  4. The label area, character positions 5 through 80, is cleared to spaces.
  5. The BEFORE BEGINNING FILE declarative is executed.
  6. VALUE OF *data-name* items are moved to the label record.
  7. The AFTER BEGINNING FILE declarative is executed.
  8. The user label is written if the label identifier equals UHL.

Open Input or I-O

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. The HDR1 label record is read.
  2. The BEFORE BEGINNING FILE declarative is executed.
  3. The VALUE OF items are verified.
  4. The AFTER BEGINNING FILE declarative is executed.
  5. The EOF1 label record is read.
  6. The file is then positioned at the beginning of the data blocks.
- If the LABEL RECORDS ARE *data-name* clause is present:
  1. The HDR1 label is read. The VALUE OF items referencing the standard label are verified.
  2. If a UHL label is present following the HDR1 label, the following process occurs:
     a. The BEFORE BEGINNING FILE declarative is executed.
     b. The VALUE OF items referencing user label items are verified.
     c. The AFTER BEGINNING FILE declarative is executed.
     d. If the file is open for I-O, the UHL label is rewritten.
  3. The EOF1 label record is read.

For RECORDING MODE IS SDF files, the beginning-of-file labels are processed by the Processor Common Input/Output System (PCIOS). No declaratives are executed for these files.

4.4.4. The READ Statement

Format 1

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

Format 2

```
READ file-name RECORD [ INTO identifier ] [WITH [ NO ] PROTECT ]
    ; INVALID KEY imperative-statement
```

Description

Format 1 is used for a file described as ACCESS MODE IS SEQUENTIAL in the FILE-CONTROL paragraph. If the ACTUAL KEY clause is specified for the file, as part of the execution of the READ, the MSCS will set the contents of the ACTUAL KEY item to the key of the current logical record.

For files designated as OPTIONAL in the FILE-CONTROL paragraph, the first execution of a READ statement will cause the AT END *imperative-statement* to be executed if no assignment has been made for the file.

The NEXT phrase can only be used with RECORDING MODE IS SDF files that are declared to be ACCESS DYNAMIC. This phrase allows the sequential retrieval of records. The execution of this READ statement causes the contents of the ACTUAL KEY *data-name* to be updated to contain the relative record number of the record made available by the READ NEXT statement. The READ statement without the NEXT phrase is for use with files that are declared to be ACCESS SEQUENTIAL. This phrase allows the sequential retrieval of records. No key value needs to be used for this format of the READ statement.

For sequentially accessed Direct files (except SDF), the AT END condition is reached when the current mass storage address of the record is equal to the mass storage end-of-file address established when the file was created, or if FILE-LIMITS were specified, when the logical end of the last segment of the file is reached and an attempt is made to read that file.

For sequentially accessed Direct SDF files, the AT END is reached when an end-of-file label record is read.

When the execution of a READ follows the execution of a START statement, the logical record associated with the contents of the ACTUAL KEY at the time of the START execution is made available.

Format 2 is used for a file described as ACCESS MODE IS RANDOM in the FILE-CONTROL paragraph.

The COBOL program must place the relative record number associated with the record to be read in the contents of the ACTUAL KEY for the file. The INVALID KEY *imperative-statement* is executed if the contents of the ACTUAL KEY are outside the FILE-LIMITS.

The WITH PROTECT option has meaning only for files named in the APPLY PROTECT clause in the I-O CONTROL paragraph. It indicates that no other cycle should be allowed to read this same record until the cycle executing this READ has completed processing of the record. Normally, processing is completed by rewriting the record back into the file; however, any subsequent access to the file by the cycle frees the record. If the cycle performing the read terminates, any record protected by that cycle is

freed. The FREE statement is used to release protection on a record when it is not to be rewritten.

The WITH NO PROTECT option has meaning only for files named in the APPLY INTERLOCK clause in the I-O CONTROL paragraph. APPLY INTERLOCK causes automatic protection of records as described above. The WITH NO PROTECT option on the READ is used to override the automatic protection.

4.4.5. The REWRITE Statement

Format

```
REWRITE record-name [ FROM identifier]

      ;INVALID KEY imperative-statement
```

Description

The REWRITE statement is used to replace a logical record on the file with a specified record. It is only valid for a file opened for I-O.

The record specified must not be longer than the original record it is replacing.

The REWRITE statement, when executed, performs all of the functions of a WRITE statement that follows a READ.

4.4.6. The START Statement

Format

```
START file-name INVALID KEY imperative-statement
```

Description

The START statement allows sequential accessing of a Direct file from a specified starting point.

The *file-name* must be defined as ACCESS IS SEQUENTIAL and can be opened for INPUT or I-O.

The relative record number at which processing is to begin must be placed in the *data-name* specified by the ACTUAL KEY clause before executing the START statement.

Normally, a file in the sequential access mode is accessed sequentially from the first record to the last or until the file is closed. If accessing is to begin at other than the first record, a START statement must be executed after the OPEN, but before the first READ statement. Processing will then continue sequentially until a START statement or a CLOSE statement is executed or until end-of-file is reached.

The INVALID KEY phrase is executed when the contents of the ACTUAL KEY are outside the FILE-LIMITS for the file or are higher than the mass storage end-of-file

address. The file is left positioned at the point prior to the execution of the START statement.

4.4.7. The USE Statement

Format 1

```
USE AFTER STANDARD ERROR PROCEDURE ON

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

Format 2

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

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

Description

The USE statement is used to specify special procedures for input, output, and input/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.

Declarative procedures are not executed for a file with a RECORDING MODE IS SDF clause.

The designated procedures are executed by the MSCS at the appropriate time as follows:

- Format 1 is executed when an unrecoverable hardware error has occurred. Exec recovery techniques were performed. The object program will be abnormally terminated after the declarative procedure has been executed.
- Format 2 is executed on OPEN and CLOSE. The declarative is executed in conjunction with a specific standard or user label.

The following general procedure is followed:

- Input or I-O file - LABEL RECORDS ARE STANDARD
    1. Read label.

2. Execute BEFORE declaratives.
3. Verify label according to the VALUE OF clause.
4. Execute AFTER declaratives.

- Output file - LABEL RECORDS ARE STANDARD
  1. Create skeleton label with default values.
  2. Execute BEFORE declaratives.
  3. Move data to label area according to VALUE OF clause.
  4. Execute AFTER declaratives.
- Input or I-O file - LABEL RECORDS ARE *data-name*
  1. Read a user label.
  2. Execute BEFORE declarative.
  3. Verify user label fields according to the VALUE OF clause.
  4. Execute AFTER declarative.
- Output file - LABEL RECORDS ARE *data-name*
  1. Move spaces to label record area.
  2. Execute BEFORE declarative.
  3. Move data to user label fields according to the VALUE OF clause.
  4. Execute AFTER declarative.
  5. Examine label identifier to determine the following:
     a. If a proper label was created, then write the label.
     b. When the label identifier is not appropriate to the condition (e.g., identifier set to UTL1 on a BEGINNING condition), then no label is written, and label processing is complete.

If the key words BEGINNING or ENDING are omitted, the designated procedures are executed for both beginning and ending labels.

FILE declaratives are only executed once for BEGINNING (if specified), and once for ENDING (if specified).

4.4.8. The WRITE Statement

Format

```
WRITE record-name [ FROM identifier-1] INVALIDKEY imperative-statement
```

Description

The WRITE statement releases a logical record to an output or I-O file.

When ACCESS IS SEQUENTIAL, the INVALID KEY clause is executed if the relative record number of the record being written is higher than the last FILE-LIMITS operand, or if the maximum mass storage assigned to the file is exceeded.

When a sequentially accessed Direct file is opened for I-O and the AT END condition has been reached, the file may be extended by executing subsequent WRITE statements.

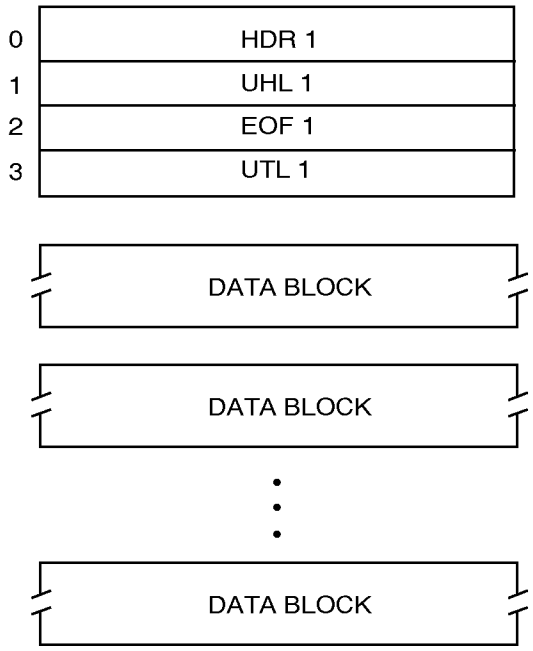
When the file is opened for I-O, all WRITE statements executed following a READ or START execution assume the meaning of a REWRITE. When a WRITE is not preceded by a READ or START, the MSCS provides positioning to the next logical record, which is then overwritten. The next logical record in this case follows the last logical record accessed by the object program, which is not necessarily the last record accessed by the cycle executing the WRITE statement.

When a WRITE statement is overwriting an original record, the length of the updated logical record can be no longer than the length of the original record.

When ACCESS IS RANDOM, the COBOL program must place in the ACTUAL KEY prior to the execution of the WRITE statement, the relative record number associated with the logical record to be written. The INVALID KEY clause is executed if the contents of the ACTUAL KEY are outside the FILE-LIMITS or if the maximum mass storage assigned to the file is exceeded.

4.5. Direct Files

4.5.1. Unlabeled File Structure



The space for system labels (HDR1/EOF1) and user labels (UHL1/UTL 1) is always allocated in the file, even if LABEL RECORDS ARE OMITTED is specified.

4.5.2. Labeled File Structure

For mass storage files (except SDF), ASCII COBOL processes a subset of the tape labels which are defined in the American National Standard documents X3.2/759, X3.2.5/128, and the revision of X3.2/513 dated July 18, 1968. These labels are 28 words in length and occupy sectors 0 through 3 of the file.

The following table gives the types of labels with their names, identifiers, and numbers. The table also shows which labels are required and which are optionally permitted.

Type	Identifier and Number	Created By	Name	Required/ Optional
Beginning of File	HDR1	System File Handler	File Header Label	Required
	UHL1	User	User File Header Label	Optional

continued

Type	Identifier and Number	Created By	Name	Required/ Optional
End of File	EOF1	System File Handler	End-of-File Label	Required
	UTL1	User	User End-of-File Label	Optional

Note: *When user label records are present, they always follow the system label of the same type.*

4.5.3. Label Record Formats

4.5.3.1. File Header Label (HDR1)

Character Positions	Length	Standard Default Values	Description	VALUE OF name
1-3	3	HDR	Label identifier	None
4	1	1	Label number	None
5-21	17	External-file-name	File identifier	FILE-ID
22-27	6	UNIVAC	Set identifier	SET-ID
28-31	4	0001	File section number	None
32-35	4	0001	File sequence number	None
36-39	4	0001	Generation number	None
40-41	2	00	Generation version number	None
42-47	6	Date when file was created	Creation date in Julian form preceded by one space	CREATION-DATE
48-53	6	Creation date plus period from ASG statement	Purge date in Julian form preceded by one space	PURGE-DATE
54	1	Space	Accessibility	FILE-ACCESS
55-60	6	000000	Block count	None
61-73	13	File qualifier from ASG statement	File qualifier	FILE-QUALIFIER

continued

Character Positions	Length	Standard Default Values	Description	VALUE OF name
74-80	7	Spaces	Unused	None
81-84	4	Mass-storage end-of-file address	Unused	None

4.5.3.2. End-of-File Label (EOF1)

The EOF1 label has the same format as the HDR1 label. The standard default values are the same except for the "block count" field, which is set to the number of physical data blocks contained in the file.

The EOF1 label block occupies all 28 words of sector 2 on mass storage. Words 21 through 27 (in other words, character positions 85-96) are used as needed for file control information. Current usage is as follows:

21 FASTRAND end-of-file address (sector address whole word FC$FEA)

22 Not used = zero

23 Creation time information

H1 - maximum block size in words (FC$BLK)

03 - actual key length in words (FC$KL)

Bits 27-29 - Key Shift flag for START verb

S6 - file type (FC$TYP)

24 Label flag

S6 - (FC$LBL)

25 Not used = zero

26 Not used = zero

27 Not used = zero

The VALUE OF clause causes the fields to be modified on output or verified on input exactly as for the HDR1 description.

4.5.3.3. User Labels

In addition to the system standard labels, mass storage files may optionally include two user labels: one UHL1 and one UTL1.

In order to create user labels on an output file, the COBOL program must include the LABELS ARE *data-name* clause. User labels are then created by the use of the VALUE OF *data-name* clause or by the specification of one or more USER LABEL PROCEDURE declarative sections or both.

The VALUE OF processing is performed each time a user label may be created. No distinction is made between references to *data-name* clauses in different label record descriptions. It is the user's responsibility to ensure that the VALUE OF items are meaningful at the time a user label may be created.

A valid user label conforms to the following description:

Character Position	Length	Name	Required Contents
1-3	3	Label identifier	UHL for user beginning file label. UTL for user end-of-file label
4	1	Label number	1
5-80	76	As defined by user	May be set to any ASCII number or character string

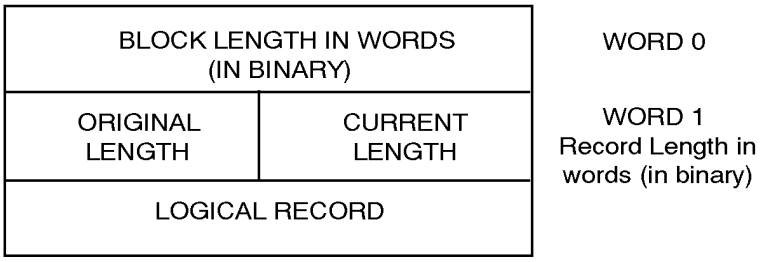
The following listing indicates the time at which it may be created (or processed on input) and the sequence of processing.

Label Identifier	Events Causing Label Processing to Occur	Sequence of Label Processing
UHL-User file header labels	OPEN	1. BEFORE BEGINNING FILE declarative is executed. 2. VALUE of processing. 3. AFTER BEGINNING FILE declarative is executed.
UTL-User end-of-file labels	CLOSE	1. BEFORE ENDING FILE declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING FILE declarative is executed.

For output files, the user label is written only if the label identifier is set correctly in the label record area. When a valid label exists, it is written.

For input files, label processing is performed only if a label with that appropriate identifier exists.

4.5.4. Data Block Format



Data blocks for non-SDF Direct files start in sector 4. Each block is allocated the number of sectors (28 words) necessary to contain the largest record in the file. In addition, two words are appended to the beginning of each record by the MSCS.

The block size can be computed as:

$$BS = \frac{MRL + 29}{28}$$

where:

BS = Block size

MRL = Number of words in largest logical record

4.6. SDF Direct Files

For a description of the file structure, label record format, and data block format for RECORDING MODE IS SDF files, see the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582, (current version).

Section 5
Indexed Sequential I-O

5.1. General

An Indexed Sequential file is a file maintained on mass storage in sequential order, but with an index structure provided such that random access (based on the indexing data as the key) may be used.

An Indexed Sequential file is composed of two elements: the data records themselves, and an indexed structure. The data records (logical records) are blocked into a physical block (called a record block) according to the BLOCK CONTAINS clause. The keys associated with each logical record are also included in the record block along with other necessary control information.

The index structure contains one entry per record block. This index entry contains the key of the last record in the record block and an address of that block. As the index structure grows to sufficient size to require multiple blocks itself, additional index levels are constructed which point to lower-level indexes (see Figure 5-1). Construction of the index structure is done when the file is initially created. Note, however, that the restriction imposed as a consequence, that an Indexed Sequential file must be created as an output Indexed Sequential file with records written in sequence prior to any other file usage.

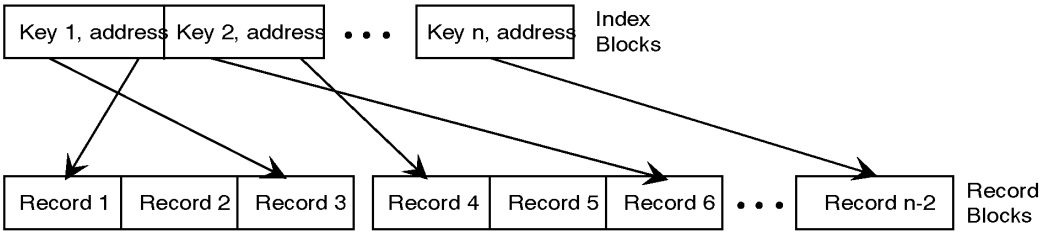


Figure 5-1. Index Record Construction

Indexed sequential processing provides for random or sequential input to the program from a previously created file. It also provides for output operations of addition of new records, modification of records already in the file, or deletion of records in the file. Modification is accomplished on a file opened as an input/output file, with the record first read in and modified, then written back to the file. Deletion is similar for an input/output file; the record is read, then deleted using the DELETE verb.

Insertion of new records may be accomplished in either of two ways. Where space has been physically reserved in the data blocks and is still available, the insertion will take

place in normal record sequence. Should such space not be available, then an overflow block is created. References to records in an overflow block require first a reference to the normal data block which then points to the overflow block; thus, each access to a record in an overflow block is somewhat more costly in time than an access to a record in the normal data block area.

Reorganization of a file is periodically desirable as insertions of records fill up the reserved space, and particularly as overflow blocks see any significant usage. Such reorganization is accomplished by reading the file sequentially and writing a new Indexed Sequential file. The recommended technique is to read the Indexed Sequential file and write the file to a tape (which then also provides file backup), followed by reading of the tape and writing into the new Indexed Sequential file.

Requirements for Indexed Sequential file usage in each of the COBOL program sections are considered in the following subsections.

5.2. Environment Division

5.2.1. The FILE-CONTROL Paragraph

5.2.1.1. The ACTUAL KEY Clause

Format

```
[ ,ACTUAL KEY IS  data-name-1 ]
```

Description

The ACTUAL KEY entry must be specified. The parameter *data-name-1* is limited to 256 words of any usage. The contents of *data-name-1* should not contain a value of binary zeros to be used as an actual key, since this value is reserved for use by the Indexed File Handler. The INVALID KEY clause will be executed if there is any attempt to read, write, or locate a key of binary zeros.

When an indexed file is OPEN for OUTPUT, the contents of *data-name-1* must be set by the COBOL program prior to the execution of a WRITE statement. Each WRITE must supply a key which is in ascending sequence with the previous key, according to a bit-by-bit compare.

When a file is opened for I-O in the sequential access mode, the above rule also applies if the file is being extended beyond its end-of-file point.

The ACTUAL KEY contents must be set by the COBOL program prior to the execution of the following statements:

1. WRITE when the file is opened for OUTPUT or is being extended.
2. All statements when ACCESS IS RANDOM.
3. START statement when ACCESS IS SEQUENTIAL.

4. START, REWRITE, WRITE, and READ (not READ NEXT) statements when ACCESS is DYNAMIC.

The ACTUAL KEY is updated by the MSCS following each READ when ACCESS IS SEQUENTIAL and each READ NEXT when ACCESS is DYNAMIC.

5.2.1.2. The ASSIGN Clause

Format

```
ASSIGN TO   implementor-name-1[,implementor-name-2]...
```

Description

Implementor-name-1 may be one of the following:

MASS-STORAGE
MASS-STORAGE-28
MASS-STORAGE-56
MASS-STORAGE-112

In the form MASS-STORAGE-*nnn*, the integer value *nnn* refers to the prepping factor of the mass storage file. These alternatives are available to allow blocking of records to be sensitive to different physical record sizes on disk devices. The block size will be calculated according to the BLOCK CONTAINS clause and then rounded up to the next multiple of the specified physical record size. With the implementor-name MASS-STORAGE, a default value of 28 words is used in rounding up block size (unless the user changes FC$PREP to 56 or 112, in which case 56 or 112 is used).

If *implementor-name-2* is specified, it is a 1- to 12-character name used to link the COBOL file to Exec file-control and should appear in an @ASG or @USE statement in the run stream for executing the object program.

When *implementor-name-2* is not specified, the first 12 characters of the *file-name* are used to link the COBOL file to the OS 1100 Exec file control.

All recursions of *implementor-name-1*, *implementor-name-2*..., are for documentation only.

5.2.1.3. The PROCESSING MODE Clause

The PROCESSING MODE clause specifies the order in which logical records are processed by the COBOL program.

Format

```
[, PROCESSING MODE IS SEQUENTIAL ]
```

Description

The PROCESSING MODE clause specifies the order in which logical records are processed by the COBOL program.

When PROCESSING MODE IS SEQUENTIAL is specified, the logical records are normally processed in the order in which they are accessed. In this case, only one area is allocated to contain a logical record. If the COBOL program has reference to such a file in a USE FOR RANDOM PROCESSING declarative section, appropriate lock techniques must be used by the programmer to ensure the integrity of the logical record if simultaneous accesses to it are possible. If this clause is absent, SEQUENTIAL PROCESSING is assumed.

5.2.1.4. The RESERVE Clause

Format

Description

The RESERVE clause is used to specify the number of buffer storage areas to be used in processing the file. When the RESERVE clause is omitted, two buffer areas are allocated. If RESERVE NO is specified, one buffer area is allocated. RESERVE *integer-2* causes *integer-2* + 1 buffer areas to be allocated. When the APPLY CORE-INDEX clause is specified in the I-O-CONTROL paragraph, an additional buffer area is allocated.

For creating an Indexed Sequential file, two buffer areas are sufficient for optimum processing.

When randomly accessing an indexed file, all buffers which are allocated are used. An attempt is made to retain as many blocks in main storage as possible, minimizing the number of input/output accesses.

Buffer areas are not compiled as part of the object program. Instead, during execution, the object program is dynamically extended to allow for the buffer areas when the file is opened. The buffer areas are released when the file is closed.

5.2.2. The I-O-CONTROL Paragraph

Function

The I-O-CONTROL paragraph provides for the protection of record areas.

Format

```
I-O-CONTROL.
    [; APPLY INTERLOCK ON file-name-1[,file-name-2]...]
    [; APPLY PROTECT ON file-name-3[FOR integer-1RECORDS]
        [,file-name-4[FOR integer-2 RECORDS ]]...]
    [;APPLY CORE-INDEX ON file-name-6[,file-name-6]...]
```

Syntax Rules

1. The I-O-CONTROL paragraph is optional.
2. The APPLY INTERLOCK clause and the APPLY PROTECT clause must not specify the same *file-name*.
3. The values *integer-1*, *integer-2*,... must be less than or equal to the value of the integer specified in the PROCESSING MODE clause. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).

General Rules

1. The APPLY INTERLOCK clause and the APPLY PROTECT clause may be specified only for files which meet the following criteria:
   a. Opened in the I-O mode.
   b. Indexed sequential file in random or dynamic access mode.
   c. The File-Control paragraph must contain the PROCESSING IS RANDOM clause.
2. Whenever a keyed record is read from a file with the APPLY INTERLOCK clause, that record is locked out until,
   a. A record is read from that file;
   b. A record is written to that file;
   c. The record is freed; or
   d. The controlling random processing cycle is complete.

   Any other program or cycle within the same COBOL run unit attempting to read a "locked out" record will be suspended until the lock is removed.
3. For a file with the APPLY INTERLOCK clause, the number of records concurrently locked for the file is equal to the number of concurrently active cycles accessing the file.
4. The parameters *integer-1*, *integer-2*,... of the APPLY PROTECT clause specify the maximum number of records to be locked out concurrently. The number of concurrently locked records may be less than the number of active cycles accessing the file.
5. The APPLY CORE-INDEX is used to specify that an extra buffer storage area be allocated to retain the file's highest-level index block in main storage.

5.3. Data Division

5.3.1. The BLOCK CONTAINS Clause

Format

```
[;BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS   }]
                                          {CHARACTERS}
```

Description

The BLOCK CONTAINS clause specifies the size of the physical record or block. The physical grouping in no way affects the logic of the program; however, it may affect the amount of mass storage needed to store data in an indexed file. With this in mind, the programmer should attempt to establish the most efficient correlation between the physical and logical record. There must be at least one record per block. Blocks may not contain partial records (in other words, records may not overlap blocks).

When this clause is used, the following rules apply:

1. The parameters *integer-1* and *integer-2* must be unsigned numeric literals.
2. If only *integer-2* is used, it represents the exact size of the physical record. If both *integer-1* and *integer-2* are used, they indicate the minimum (*integer-1*) and maximum (*integer-2*) size of the physical record. The compiler will specify blocking such that the maximum size will be used.
3. When the CHARACTERS phrase is used, the compiler will determine the word size of the block from the number of characters specified. It will also ensure that the number of characters specified is large enough to contain one logical record of the maximum size, plus the actual key and the required control information. When specified, *integer-1* is used to determine the maximum number of words to be placed in the block when the file is created.
4. When the RECORDS phrase is used, the compiler will determine the block size which can contain *integer-2* logical records of maximum size plus key size, plus additional space for required control words. When specified, *integer-1* is multiplied by the maximum logical record size to determine the maximum number of words to be placed in the block when the file is created.

When the clause is omitted, the block size will be computed to contain one logical record of maximum size plus one key and additional control information.

5.3.2. The LABEL RECORDS Clause

Format

```
;LABEL { RECORD IS  } {STANDARD                                  }
       { RECORDS ARE} {OMITTED                                   }
                      {data-name-1[,data-name-2]...              }
```

Description

Label records are located on the first four sectors of mass storage allocated to the file. The LABEL RECORDS clause permits the identification of these label records. The LABEL clause must appear for each file description.

If the user desires only the system standard label records, then the STANDARD option may be used. The OMITTED phrase specifies that the COBOL program does not require labels for the file.

The parameter *data-name-1* is the name of the label record which must be the subject of a Record Description entry associated with the file. The *data-name* must not appear in the DATA RECORDS clause of the File Description.

When user label records are specified (*data-name-1*, *data-name-2*...), the user may define fields in the description of *data-name-1* (up to 80 characters) and have values placed in them (OUTPUT) and checked for (INPUT and I-O) by using the VALUE clause. All fields described in label records must have DISPLAY usage. Such fields may not be referred to outside of the USE for LABEL declarative sections in the Procedure Division.

5.3.3. The RECORDING MODE Clause

Format

```
[                                  { BLANK [SIGN] }  ]
[ ;RECORDING MODE IS               { COMPACT      }  ]
[                                  { INTERNAL     }  ]
[                                  { SIGN         }  ]
```

Description

The RECORDING MODE clause specifies for mass storage files the format of the logical records on mass storage which may be different from the format in main storage. Any appropriate data conversions are performed as data records are transferred between main storage and mass storage.

When RECORDING MODE IS INTERNAL is specified, it means that all records are read or written exactly as they appear in main storage with no conversion taking place. If a RECORDING MODE clause is not present, INTERNAL is assumed.

If a RECORDING MODE other than INTERNAL is specified and the file contains multiple data record descriptions, a record selector field within each record description is required. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version) for a description of record selector fields.

When the record is read or written, the selector is tested to determine which record conversion must be performed.

5.3.3.1. The RECORDING MODE IS BLANK Clause

When the RECORDING MODE IS BLANK clause is stated, leading blanks in numeric DISPLAY or DISPLAY-1 fields are converted to leading zeroes as the records are read from mass storage.

5.3.3.2. The RECORDING MODE IS BLANK SIGN Clause.

The RECORDING MODE IS BLANK SIGN clause means that the conversions for both RECORDING MODE IS BLANK and RECORDING MODE IS SIGN are to be performed on the file.

5.3.3.3. The RECORDING MODE IS COMPACT Clause

The RECORDING MODE IS COMPACT clause means that the records on the file contain variable length arrays. Each record on mass storage is compressed and is automatically converted to or from expanded form in main storage. The DEPENDING phrase of the OCCURS clause is used whenever a variable number of occurrences of a dimensioned item is desired. If a DEPENDING phrase is used within a record description of a file whose RECORDING MODE IS COMPACT, the value of a data item is used to specify the conversion required such that unused occurrences do not appear in the external media.

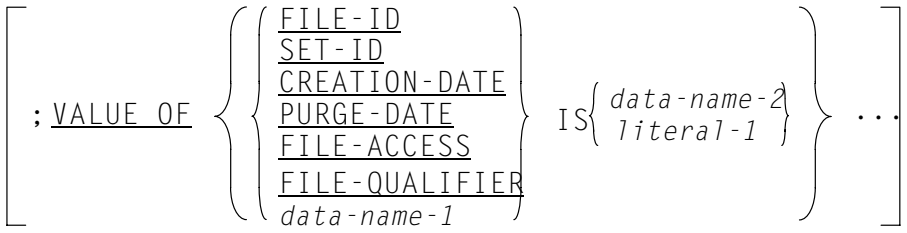
If a RECORDING MODE IS COMPACT file specifies the APPLY EXDEF clause and the file contains a forward referenced OCCURS DEPENDING ON data item, the APPLY EXDEF clause can only be used in the main program.

5.3.3.4. The RECORDING MODE IS SIGN Clause

The RECORDING MODE IS SIGN clause indicates that every signed numeric DISPLAY or DISPLAY-1 data item is given an overpunched sign on the low-order digit as it is read into main storage. On mass storage, the sign is carried as an explicit character preceding the high-order digit of the field.

5.3.4. The VALUE OF Clause

Format



Description

The VALUE OF clause specifies the value of an item in a label record. This clause should not be present for a file when LABEL RECORDS ARE OMITTED is specified.

The following names are names of fields in the system standard labels and have the implied definition of:

	Field Name	Implied Definition
02	FILE-ID	PIC X(17)
02	SET-ID	PIC X(6)
02	CREATION-DATE	PIC 9(6)
02	PURGE-DATE	PIC 9(6)
02	FILE-ACCESS	PIC X(1)
02	FILE-QUALIFIER	PIC X(12)

In the absence of any or all of the above names in the VALUE OF clause when a file is opened for output, the following values will be assumed:

FILE-ID

Will be set to the 12-character external file name.

SET-ID

Will be set to UNIVAC.

CREATION-DATE

Will be set to the current date.

PURGE-DATE

Will be set to the current date plus the expiration period specified on the @ASG statement or system default if no expiration period is specified.

FILE-ACCESS

Will be set to a space indicating unlimited access to the file.

FILE-QUALIFIER

Will be set to the file qualifier.

In addition to the above system standard label names, the VALUE OF *data-name-1* form may be used provided LABEL RECORDS ARE *data-name* is also specified. In this case, *data-name-1*, etc., must be defined in the record description for a label record.

The contents of *data-name-2* or *literal-1* will be moved to the label field on output or checked against the label field on input. The parameter *data-name-2* must be described as USAGE DISPLAY (ASCII).

When user label records are specified. the VALUE OF processing takes place in two steps:

1. When the system standard label records are being read or written, the VALUE OF standard label names are checked or moved, respectively.
2. When user label records are being read or written, the VALUE OF *data-name* clauses are checked or moved, respectively.

5.3.4.1. The VALUE OF CREATION-DATE Clause

The contents of this field are set to the current date at the time the file is opened for output. This clause modifies the contents of the field to the value specified. The value field is written in the COBOL program in the form *MMDDYY* and is converted to Julian form when the HDR1 label is created. At this time, the contents of the month, day, and year values are verified for a proper range of numbers.

5.3.4.2. The VALUE OF *data-name* Clause

In addition to the system standard labels, mass storage files may optionally include two user labels: one UHL1 and one UTL1.

In order to create user labels on an output file, the COBOL program must include the LABELS ARE *data-name* clause. User labels are then created by the use of the VALUE OF *data-name* clause, by the specification of one or more USER LABEL PROCEDURE declarative sections, or both.

The VALUE OF processing is performed each time a user label may be created. No distinction is made between references to *data-name* clauses in different label record descriptions. It is the user's responsibility to ensure that the VALUE OF items are meaningful at the time a user label may be created.

The following chart indicates the time at which a user label may be created (or processed on input) and the sequence of processing.

Label Identifier	Events Causing Label Processing to Occur	Sequence of Label Processing
UHL-User file header labels	OPEN	1. BEFORE BEGINNING FILE declarative is executed. 2. VALUE of processing. 3. AFTER BEGINNING FILE declarative is executed.
UTL-User end-of-file labels	CLOSE	1. BEFORE ENDING FILE declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING FILE declarative is executed.

For output files, the user label is written only if the label identifier is set correctly in the label record area. When a valid label exists, it is written.

For input files, label processing is performed only if a label with that appropriate identifier exists.

5.3.4.3. The VALUE OF FILE-ACCESS Clause

One space is moved to the FILE-ACCESS field when the HDR1 is created. The VALUE OF FILE-ACCESS clause may modify the contents of the field. A space represents unlimited access to the file.

5.3.4.4. The VALUE OF FILE-ID Clause

If specified by the VALUE OF FILE-ID clause, the file identifier is checked by the MSCS on files opened for input or I-O and moved by the file handler on output files.

When creating a file, the external *file-name* is moved to the skeleton label. The VALUE OF FILE-ID clause may be used to modify that file name.

5.3.4.5. The VALUE OF FILE-QUALIFIER Clause

The contents of the FILE-QUALIFIER field are set originally to the qualifier specified in the @ASG statement, or the qualifier specified in the currently active @QUAL statement, or by default to the project field in the @RUN statement. The VALUE OF FILE-QUALIFIER clause modifies the contents of the field to the value specified.

5.3.4.6. The VALUE OF PURGE-DATE Clause

The contents of the PURGE-DATE field are set to the current date at the time the file is opened for output plus the number of days to expire specified in the @ASG statement or system default. The VALUE OF PURGE-DATE clause modifies the contents of the field to the value specified. The rules for the value for PURGE-DATE are the same as for CREATION-DATE.

The PURGE-DATE field represents the day on which the file may be overwritten.

5.3.4.7. The VALUE OF SET-ID Clause

The COBOL program may set the contents of this field without restriction.

5.4. Procedure Division

5.4.1. The CLOSE Statement

Format

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

Description

The CLOSE statement terminates processing of one or more files. The CLOSE statement causes end-of-file label processing to be performed and the buffer areas to be released.

For a file opened for OUTPUT, the CLOSE statement causes the final index block to be written. The initial overflow block is allocated at this time.

If a file has been specified as OPTIONAL and is not present, the standard end-of-file processing is bypassed.

When the WITH LOCK phrase is used, this file may not be reopened subsequently in the same execution.

For files described as LABELS ARE OMITTED, the MSCS creates a standard EOF1 label containing system default values. File control information is inserted after the EOF1 label in the I-O buffer. The 28-word I-O buffer containing the EOF1 label is then written.

For labeled files. the following label processing occurs:

- Input or I-O file, LABEL RECORDS ARE STANDARD
  1. The BEFORE ENDING FILE declarative is executed.
  2. The EOF1 label is read.
  3. The VALUE OF items are verified.
  4. The AFTER ENDING FILE declarative is executed.

5. For files opened for I-O, the EOF1 label record is rewritten.

- Input or I-O file, LABEL RECORDS ARE *data-name*
  1. The EOF1 label is read.
  2. The VALUE OF items referencing the standard label names are verified.
  3. For files opened for I-O the EOF1 label record is rewritten.
  4. The user ending label is read. If the label identifier contains UTL, the following occurs:
     a. The BEFORE ENDING FILE declarative is executed.
     b. The VALUE OF items referencing user label names are verified.
     c. The AFTER ENDING FILE declarative is executed.
     d. For files opened for I-O, the EOF1 label record is rewritten.
- Output file, LABEL RECORDS ARE STANDARD
  1. An EOF1 label record is created with standard default values.
  2. The BEFORE ENDING FILE declarative is executed.
  3. The VALUE OF items are moved to the label record.
  4. The AFTER ENDING FILE declarative is executed.
  5. File control information is inserted after the EOF1 label record in the I-O buffer.
  6. The 28-word I-O buffer containing the EOF1 label record is written.
- Output file, LABEL RECORDS ARE *data-name*
  1. An EOF1 label record is created with the standard default value.
  2. The VALUE OF items referencing standard label names are moved to the label record.
  3. File control information is inserted after the EOF1 label record in the I-O buffer.
  4. The 28-word I-O buffer containing the EOF1 label record is written.
  5. The label record area, character positions 5 through 80, is cleared to spaces.
  6. The BEFORE ENDING FILE declarative is executed.
  7. The VALUE OF items referencing user label names are moved to the label record.
  8. The AFTER ENDING FILE declarative is executed.
  9. The user ending label record is written.

5.4.2. The DELETE Statement

Format

```
DELETE record-name; INVALIDKEY imperative-statement
```

Description

The DELETE statement causes a logical record to be removed from the file.

The DELETE statement is valid only for files which are opened for I-O.

When the file is described as ACCESS is RANDOM or ACCESS is DYNAMIC. the key value associated with the logical record to be deleted must be placed in the *data-name* referenced by the ACTUAL KEY clause prior to the execution of the DELETE statement. If the MSCS can locate no record on the file with a matching key, the INVALID KEY *imperative-statement* is executed.

When a file is described as ACCESS IS SEQUENTIAL, the MSCS determines the logical record to be deleted based on the previous statement executed for the file. When a DELETE follows a READ, the record which was read is deleted. When a DELETE follows a START, the record associated with the key value supplied in the ACTUAL KEY for the START statement is deleted. A DELETE following the execution of a WRITE, REWRITE, or another DELETE causes the file handler to position to the next sequential logical record, which is then marked as deleted. When a DELETE statement is executed and the file is positioned beyond its end-of-file, the INVALID KEY *imperative-statement* is executed.

When a logical record is deleted, the value of the current locator for the record is set to binary zeros. The logical record itself is removed from the file, making spare in the record block available for subsequent inserts.

5.4.3. The FREE Statement

The FREE statement releases a record for access by other activities.

Format

```
FREE file-name RECORD
```

Syntax Rule

The FREE statement may be specified only for files which were named in the APPLY INTERLOCK or APPLY PROTECT clause in the I-O-CONTROL paragraph (see 5.2.2).

General Rules

1. The associated file must be open in the I-O mode at the time of the execution of this statement (see 5.4.4).
2. When a record is read with protection, any other program or cycle attempting to read that record is suspended until the protection lock is removed. Normally, the protection is removed by rewriting the record or reading another record from the file. The FREE statement is used to remove the lock prior to another access to the file, so that any other cycle waiting to read that record can be resumed.

3. Following the execution of the last statement in a random processing declarative section, any records which have been read with protection are released by the file handler.
4. The value of the key associated with an indexed sequential file is not used for the FREE statement.
5. The logical record continues to be available to the COBOL program following the execution of a FREE statement.

5.4.4. The OPEN Statement

Format

```
OPEN    { INPUT  file-name-1 [ ,file-name-2]...  }
        { OUTPUT file-name-3 [ ,file-name-4]...  }  ...
        { I-O    file-name-5 [ ,file-name-6]...  }
```

Description

The OPEN statement initiates processing of the named files by checking or writing labels and performing any other input/output operations necessary prior to accessing the first record in a given file. However, the OPEN statement does not obtain or release the first data record; a READ or WRITE statement must be executed.

If an input file is designated as OPTIONAL and ACCESS IS SEQUENTIAL in the FILE-CONTROL paragraph, the MSCS interrogates for the presence of the file. If the file is not present, the first READ statement for the file causes the imperative statement in the AT END phrase to be executed.

The following label processing occurs when an indexed file is opened:

Open Output

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. The HDR1 label is created with standard values.
  2. The BEFORE BEGINNING FILE declarative is executed.
  3. The VALUE OF processing is performed.
  4. The AFTER BEGINNING FILE declarative is executed.
  5. The HDR1 label record is written.
- If the LABEL RECORDS ARE *data-name* clause is present:
  1. The HDR1 label is created with standard values.
  2. The VALUE OF items referencing the standard label are moved to the label area.
  3. The HDR1 label is written.
  4. The label area, character positions 5 through 80, is cleared to spaces.
  5. The BEFORE BEGINNING FILE declarative is executed.
  6. VALUE OF *data-name* items are moved to the label record.

7. The AFTER BEGINNING FILE declarative is executed.
8. The user label is written if the label identifier equals UHL.

Open Input or I-O

- If the LABEL RECORDS ARE STANDARD clause is present:
  1. The HDR1 label record is read.
  2. The BEFORE BEGINNING FILE declarative is executed.
  3. The VALUE OF items are verified.
  4. The AFTER BEGINNING FILE declarative is executed.
  5. The EOF1 label record is read.
  6. The input or I-O file's maximum block size and actual key length are verified by comparing them to the corresponding fields in EOF1.
- If the LABEL RECORDS ARE *data-name* clause is present:
  1. The HDR1 label record is read.
  2. The VALUE OF item referencing the standard label is verified.
  3. The UHL1 label record is read.
  4. If the label identifier equals UHL:
     a. The BEFORE BEGINNING FILE declarative is executed.
     b. The VALUE OF items referencing user label items are verified.
     c. The AFTER BEGINNING FILE declarative is executed.
     d. If the file is opened for I-O, the UHL label is rewritten.
  5. The EOF1 label record is read.
  6. The input or I-O file's maximum block size and actual key length are verified by comparing them to the corresponding fields in EOF1.

5.4.5. The READ Statement

Format 1

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

Format 2

```
READ file-name RECORD [INTO identifier] [WITH [NO] PROTECT]
    ; INVALID KEY imperative-statement
```

Description

Format 1 is used for a file described as ACCESS MODE IS SEQUENTIAL or ACCESS MODE IS DYNAMIC in the FILE-CONTROL paragraph. The NEXT phrase may only be

used if ACCESS MODE is DYNAMIC has been stated. As part of the execution of the READ, the MSCS will set the contents of the ACTUAL KEY item to the key of the current logical record.

For files designated as OPTIONAL in the FILE-CONTROL paragraph, the first execution of a READ statement will cause the AT END *imperative-statement* to be executed if no assignment has been made for the file.

If the NEXT phrase of the READ statement is used for a file which has been described as ACCESS MODE IS DYNAMIC, the result will be the same as a READ of an ACCESS MODE IS SEQUENTIAL file.

The READ which contains the NEXT phrase must be preceded logically by one of the following:

READ
READ NEXT
OPEN INPUT
OPEN I-O
START

Upon successful completion of one of the preceding instructions, a READ NEXT statement will retrieve the next logical record of the file, or take the AT END path. Intervening WRITE or REWRITE statements do not affect the file positioning.

If the INVALID KEY or AT END conditions exist, one of the following sequences must be successfully executed before issuing a READ NEXT statement:

CLOSE/OPEN
READ INVALID KEY
START

Format 2 is used for a file described as ACCESS MODE IS RANDOM or ACCESS MODE IS DYNAMIC in the FILE-CONTROL paragraph.

The COBOL program must place the value of the key associated with the record to be read in the contents of the ACTUAL KEY for the file. The INVALID KEY *imperative-statement* is executed if no record exists on the file with a matching key.

The WITH PROTECT phrase has meaning only for *file-name* phrases in the APPLY PROTECT clause in the I-O-CONTROL paragraph (see 5.2.2). It indicates that no other cycle should be allowed to read this same record until the cycle executing this READ has completed processing of the record. Normally, processing is completed by rewriting the record back into the file; however, any subsequent access to the file by the cycle "frees" the record. If the cycle performing the read terminates, any record protected by that cycle is freed. The FREE verb is used to release protection on a record when it is not to be rewritten.

The WITH NO PROTECT option has meaning only for files named in the APPLY INTERLOCK clause in the I-O-CONTROL paragraph. APPLY INTERLOCK causes automatic protection of records as described previously. The WITH NO PROTECT option on the READ is used to override the automatic protection.

Format 2 of the READ statement causes the index blocks to be searched for the record block containing the specified key. The record block is then read and searched for the proper record associated with the key. This might cause an overflow block to be read if the record was inserted following the original creation of the file.

5.4.6. The REWRITE Statement

Format

```
REWRITE record-name [ FROM identifier]

      ;INVALID KEY imperative-statement
```

Description

The REWRITE statement is used to replace a logical record on the file with a specified record. It is only valid for a file opened for I-O.

The record specified must not be longer than the original record it is replacing.

The REWRITE statement is executed in the same manner as a WRITE statement following a READ.

5.4.7. The START Statement

Format

```
START file-name [ KEY { IS EQUAL TO      } data-name ]
                      { iS =             }
                      { IS GREATER THAN  }
                      { IS >             }
                      { ISNOT LESSTHAN   }
                      { IS NOT <         }
```

Description

The START statement provides a basis for logical positioning within an indexed file for subsequent sequential retrieval of records.

The file must be described as ACCESS IS SEQUENTIAL or ACCESS is DYNAMIC and be opened for INPUT or I-O.

If the KEY phrase is specified, *data-name* may reference a data item specified as the ACTUAL KEY associated with *file-name*, or it may reference any data item of category alphanumeric subordinate to the *data-name* of a data item specified as the ACTUAL KEY associated with *file-name* whose leftmost character position corresponds to the leftmost character position of the ACTUAL KEY data item.

If the KEY phrase is not specified, the relational operator “IS EQUAL TO” is implied and the ACTUAL KEY field is the implied *data-name*.

The type of comparison specified by the relational operator in the KEY phrase occurs between the key associated with a record in the file and *data-name*. The comparison is binary proceeding from the leftmost position and comparing as though the key associated with the file were truncated to the size of *data-name*.

5.4.8. The USE Statement

Format 1

```
USE AFTER STANDARD ERROR PROCEDURE ON
    { file-name-1 [ ,file-name-2 ] ...
    { INPUT
    { OUTPUT
    { I-O
```

Format 2

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

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

Description

The USE statement is used to specify special procedures for input, output, and input/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 COBOL MSCS at the appropriate time as follows:

- Format 1 is executed when an unrecoverable hardware error has occurred. Exec system recovery techniques were performed. The object program will be abnormally terminated after the declarative procedure has been executed.
- Format 2 is executed on OPEN and CLOSE. The declarative is executed in conjunction with a specific standard or user label.

The following general procedure is followed:

- Input or I-O file - LABEL RECORDS ARE STANDARD
    1. Read label.

2. Execute BEFORE declaratives.
3. Verify label according to the VALUE OF clause.
4. Execute AFTER declaratives.

- Output file - LABEL RECORDS ARE STANDARD
  1. Create skeleton label with default values.
  2. Execute BEFORE declaratives.
  3. Move data to label area according to VALUE OF clause.
  4. Execute AFTER declaratives.
  5. Write the label.
- Input or I-O file - LABEL RECORDS ARE *data-name*
  1. Read a user label.
  2. Execute BEFORE declarative.
  3. Verify user label fields according to the VALUE OF clause.
  4. Execute AFTER declarative.
- Output file - LABEL RECORDS ARE *data-name*
  1. Move spaces to label record area.
  2. Execute BEFORE declarative.
  3. Move data to user label fields according to the VALUE OF clause.
  4. Execute AFTER declarative.
  5. Examine label identifier to determine the following:
     a. If a proper label was created, then write the label.
     b. When the label identifier is not appropriate to the condition (e.g., identifier set to UTL1 on a BEGINNING condition), then no label is written, and label processing is complete.

If the key words BEGINNING or ENDING are omitted, the designated procedures are executed for both beginning and ending labels.

FILE declaratives are only executed once for BEGINNING (if specified), and once for ENDING (if specified).

5.4.9. The WRITE Statement

Format

```
WRITE record-name [ FROM identifier-1] :INVALIDKEY imperative-statement
```

Description

The WRITE statement releases a logical record to an output or I-O file.

When the file is opened for OUTPUT, the contents of *data-name* referenced by the ACTUAL KEY clause must be set by the COBOL program to the value associated with the logical record. Each WRITE which is executed must supply a key which is higher in binary value than the preceding key. When a key is less than or equal to the preceding key, the INVALID KEY *imperative-statement* is executed.

If the maximum mass storage area allocated on the @ASG statement is exhausted and another write is attempted, an error diagnostic will be issued and the job aborted.

The index blocks are created by the MSCS as the record blocks become filled with logical records. The index blocks are never modified once the file has been created.

When a sequential access file is opened for I-O, the MSCS provides the ACTUAL KEY for each record which is read. When the WRITE statement is executed following a READ and implies a REWRITE, the contents of the ACTUAL KEY should not be modified by the COBOL program.

When a sequential access file is opened for I-O and the AT END condition has been reached, the file may be extended by executing subsequent WRITE statements. In this case, the rules for files opened for OUTPUT must be followed.

When a sequential access file is opened for I-O, all WRITE statements executed following a READ execution assume the meaning of a REWRITE. When a WRITE is preceded by a WRITE, REWRITE, or DELETE, the MSCS provides positioning to the next logical record, which is then overwritten. The next logical record in this case follows the last logical record accessed by the object program, not necessarily the last record accessed by the cycle executing the WRITE statement. A WRITE preceded by an OPEN or START is an INVALID KEY condition.

A REWRITE in a random processing declarative for a sequentially accessed file always causes the rewriting of the record last read by that cycle, even though subsequent reads by other cycles have caused the file to be positioned beyond the record to be rewritten.

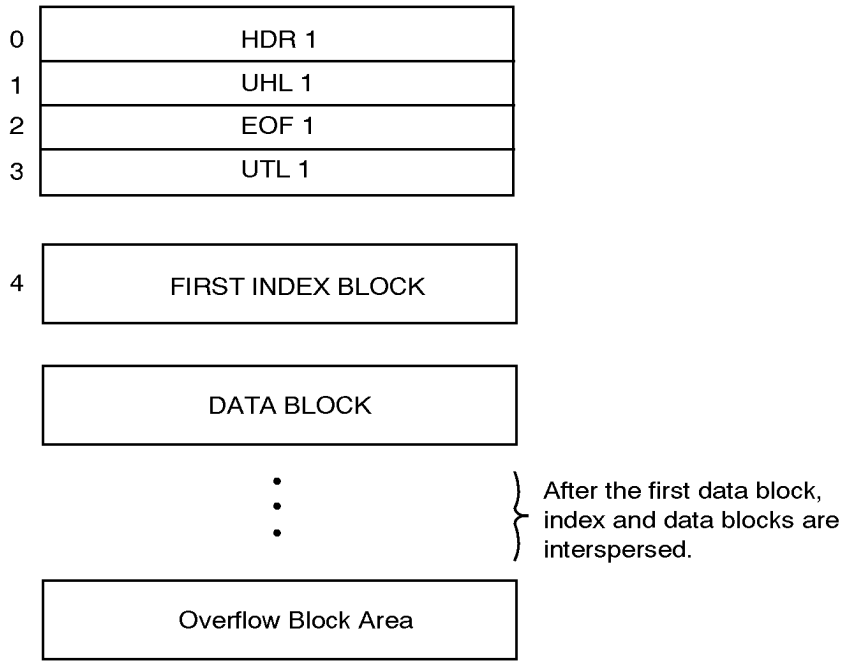
When a randomly accessed file is open for I-O, the COBOL program must supply the value of the key in the contents of the ACTUAL KEY. The MSCS uses the index blocks to locate the key. If a record is found with a matching key, the logical record is overwritten with the current record.

If a matching key is not found, the record is an insertion. An attempt is made to move the insert record into the record block for which it is in order. If no room is available, the logical record is moved into an overflow block, which is then chained into the record block which contains the next lower key.

In all cases above, when a WRITE statement is overwriting an original record, the length of the updated logical record can be no longer than the length of the original record.

5.5. Indexed Sequential Files

5.5.1. Unlabeled File Structure



Note: *Data blocks, index blocks, and overflow blocks are all the some size.*

The space for system labels (HDR1/EOF1) and user labels (UHL1/UTL1) is always allocated in the file, even if LABEL RECORDS ARE OMITTED is specified.

As for all mass storage files, sectors 0 through 3 are reserved for label blocks, sector 4 contains the first lowest level index block, followed by the first record block. The rest of the file consists of interspersed index and record blocks.

When a file is originally created, one overflow block is allocated. As the file is expanded in later processing, further overflow blocks are allocated as necessary.

5.5.2. Labeled File Structure

For mass storage files, ASCII COBOL processes a subset of the tape labels which are defined in the American National Standard documents X3.2/759, X3.2.5/128, and the revision of X3.2/513, dated July 18, 1968. These labels are 28 words in length and occupy sectors 0 through 3 of the file.

The following table gives the types of labels with their names, identifiers, and numbers. The table also shows which labels are required and which are permitted as options.

Type	Identifier and Number	Created By	Name	Required/ Optional
Beginning of File	HDR1	System File Handler	File Header Label	Required
	UHL1	User	User File Header Label	Optional
End of File	EOF1	System File Handler	End-of-File Label	Required
	UTL1	User	User End-of-File Label	Optional

Note: *When user label records are present, they always follow the system label of the same type.*

5.5.3. Label Record Formats

5.5.3.1. File Header Label (HDR1)

Character Positions	Length	Standard Default Values	Description	VALUE OF name
1-3	3	HDR	Label identifier	None
4	1	1	Label number	None
5-21	17	External-file-name	File identifier	FILE-ID
22-27	6	UNIVAC	Set identifier	SET-ID
28-31	4	0001	File section number	None
32-35	4	0001	File sequence number	None
36-39	4	0001	Generation number	None
40-41	2	00	Generation version number	None
42-47	6	Date when file was created	Creation date in Julian form preceded by one space	CREATION-DATE
48-53	6	Creation date plus period from ASG statement	Purge date in Julian form preceded by one space	PURGE-DATE
54	1	Space	Accessibility	FILE-ACCESS
55-60	6	000000	Block count	None
61-73	13	File qualifier from ASG statement	File qualifier	FILE-QUALIFIER

continued

Character Positions	Length	Standard Default Values	Description	VALUE OF name
74-80	7	Spaces	Unused	None
81-84	4	Mass-storage end-of-file address	Unused	None

5.5.3.2. End-of-File Label (EOF1)

The EOF1 label has the same format as the HDR1 label. The standard default values are the same except for the "block count" field, which is set to the number of physical data blocks contained in the file.

The EOF1 label block occupies all 28 words of sector 2 on mass storage. See the OS 2200 Processor Common Input/Output System (PCIOS) Administration and Programming Reference Manual, 7831 0588 (current version) for a more complete description of the EOF1 label.

5.5.3.3. User Labels

In addition to the system standard labels, mass storage files may optionally include two user labels; one UHL1 and one UTL1.

To create user labels on an output file, the COBOL program must include the LABEL RECORDS ARE *data-name* clause. User labels are then created by the use of the VALUE OF *data-name* clause, by the specification of one or more USE...LABEL PROCEDURE declarative sections, or both.

The VALUE OF processing is performed each time a user label may be created. No distinction is made between references to *data-name* clauses in different label record descriptions. It is the user's responsibility to ensure that the VALUE OF items are meaningful at the time a user label may be created.

A valid user label conforms to the following description:

Character Position	Length	Name	Required Contents
1-3	3	Label identifier	UHL for user beginning file label. UTL for user end-of-file label
4	1	Label number	1
5-80	76	As defined by user	May be set to any ASCII number or character string

The following chart indicates the time at which it may be created (or processed on input) and the sequence of processing.

Label Identifier	Events Causing Label Processing to Occur	Sequence of Label Processing
UHL-User file header labels	OPEN	1. BEFORE BEGINNING FILE declarative is executed. 2. VALUE of processing. 3. AFTER BEGINNING FILE declarative is executed.
UTL-User end-of-file labels	CLOSE	1. BEFORE ENDING FILE declarative is executed. 2. VALUE OF processing. 3. AFTER ENDING FILE declarative is executed.

For output files, the user label is written only if the label identifier is set correctly in the label record area, when a valid label exists, it is written.

For input files, label processing is performed only if a label with that appropriate identifier exists.

5.5.4. Data Block, Overflow Block, and Index Block Format

For a description of data block, overflow block, and index block formats, see the OS 1100 Processor Common Input/Output System (PCIOS) Administration and Programming Reference Manual, 7831 0588 (current version).

5.5.5. File Size Considerations

For a description of file size considerations, see the OS 1100 Processor Common Input/Output System (PCIOS) Administration and Programming Reference Manual, 7831 0588 (current version).

5.6. ACTUAL KEY Updating

The following chart indicates the MSCS's usage of the ACTUAL KEY data item associated with the file:

Verb	ACCESS SEQUENTIAL	ACCESS RANDOM	ACCESS DYNAMIC
READ	File handler updates the ACTUAL KEY contents with the key associated with the next sequential logical record.	COBOL program must set the ACTUAL KEY contents prior to the execution of the READ statement.	COBOL program must set the ACTUAL KEY contents prior to the execution of the READ statement.
READ NEXT	READ NEXT is invalid for sequential access.	READ NEXT is invalid for random access.	File handler updates the ACTUAL KEY contents with the key associated with the next sequential logical record.
WRITE or REWRITE	1. OPEN for OUTPUT: the COBOL program must set the ACTUAL KEY contents prior to the execution of the WRITE statement. Keys must be in ascending order. 2. OPEN for I-O: if file is being extended beyond its end-of-file, rules under A apply; otherwise the MSCS determines the logical record to be overwritten.	COBOL program must set the ACTUAL KEY contents prior to the execution of the statement. If the file is OPEN for OUTPUT, the keys must be in ascending order.	COBOL program must set the ACTUAL KEY contents prior to the execution of the statement. If the file is OPEN for OUTPUT, the keys must be in ascending order.
START	The COBOL program must set the ACTUAL KEY contents prior to the execution of the START statement.	START is invalid for random access.	The COBOL program must set the ACTUAL KEY contents prior to the execution of the START statement.
START with KEY Phrase Specified	The COBOL program must set the contents of the *data-name* specified prior to execution of the START statement.	START is invalid for random access.	The COBOL program must set the contents of the *data-name* specified prior to execution of the START statement.
DELETE	The MSCS determines the logical record to be deleted based on the position in the file. The contents of the ACTUAL KEY are not used.	The COBOL program must set the contents of the ACTUAL KEY prior to execution of the DELETE statement.	The COBOL program must set the contents of the ACTUAL KEY prior to execution of the DELETE statement.
FREE	FREE is invalid for sequential access.	The contents of the ACTUAL KEY are not used by the MSCS for the FREE statement.	The contents of the ACTUAL KEY are not used by the MSCS for the FREE statement.

Section 6
Report Writer

6.1. The Special-Names Paragraph in the Environment Division

Format

```
[ .2-character-alphanumeric-literal IS mnemonic-name]
```

Description

Mnemonic-name is used in the CODE clause in the Report Description when a 2-character literal is desired as the code.

6.2. The Report Description Entry in the Data Division

Function

The CODE clause specifies a 2-character literal that identifies each print line as belonging to a specific report.

Format

```
[ ; CODE  mnemonic-name]
```

Syntax Rules

1. *Mnemonic-name* must be associated with two identifying characters in the SPECIAL-NAMES paragraph of the Environment Division.
2. If the CODE clause is specified for any report in a file, then it must be specified for all reports in the same file.

General Rules

1. When the CODE clause is specified, the two characters associated with the mnemonic-name are placed in the first two character positions of each Report Writer logical record.
2. The positions occupied by the two characters associated with the *mnemonic-name* are not included in the description of the print line but are included in the logical record size.
3. The clause is ignored if the report file is assigned to the printer.

Section 7
Library

7.1. The COPY Statement

Function

The COBOL library contains text available to the source program at compilation time through the use of the COPY statement. The effect of the compilation of library entries is the same as if the text were actually written as part of the source program.

When the E option is present in the Extra Options field of the @ACOB processor call statement, the COBOL library may contain three types of entries as follows:

- Entries for the Environment Division consisting of equipment-oriented information.
- Entries for the Data Division consisting of information pertaining to file, communication, and data description entries.
- Entries for the Procedure Division consisting of sequences of procedure paragraphs and sections.

Format

```
COPY
    [ REPLACING word-1 BY { word-2      }
                          { identifier-1}
                          { literal-1   }

        [ ,word-3 BY { word-4      } ]  ... ]
                     { identifier-2 }
                     { literal-2    }
```

Description

The COPY statements permit the incorporation of existing library text entries into the Environment, Data, and Procedure Divisions. A library text entry is a segment of COBOL source language. By specifying the appropriate *text-name* within an appropriate format, the programmer can cause an entry to be copied from the library during compilation. and the result is the same as if the entry had been written as part of the source program.

Text-name must be in the form of an Exec procedure name. Up to 30 characters can be used.

In this format, *word* represents any of the following:

- data-name
- procedure-name
- condition-name
- mnemonic-name
- file-name
- saved area-name
- communication description-name
- report-name

No other statement or clause may appear in the same entry as the COPY statement. The library text to be copied must not contain any COPY statements and must contain COBOL source language that is syntactically correct. The copying process is terminated automatically when the END statement of the procedure is reached.

If the REPLACING option is used, each of the library words or identifiers specified in the format is replaced by the stipulated word or identifier with which it is associated in the format. On the source listing, all copied material will be realigned to allow for word or identifier replacement of greater length. This replacement does not alter the material as it exists in the library, and the entry may be called again in the same program with different replacements. Words specified in the REPLACING option may be any COBOL word. The replacement by an *identifier* includes the replacement of all associated qualifiers, subscripts, and indexes.

The word replacement within the Procedure Division will replace procedure-names as well as data-names, file-names, etc.

7.2. Library Entries for the Environment Division

There are five types of entries in the library that may be associated with the Environment Division: entries for the SPECIAL-NAMES, SOURCE-COMPUTER, OBJECT-COMPUTER, FILE-CONTROL, and I-O-CONTROL paragraphs.

To use an entry contained in the COBOL library, the COPY clause must follow the appropriate paragraph-name and indicate the procedure-name of the entry to be copied from the library. The formats are as follows:

```
SOURCE-COMPUTER      copy-statement.

OBJECT-COMPUTER.     copy-statement.

SPECIAL-NAMES.       copy-statement.

FILE-CONTROL.        copy-statement.

I-O-CONTROL.         copy-statement.
```

Environment Division library text may begin with:

1. An Environment Division paragraph-name.
2. A comment statement.
3. Syntactically valid entries associated with the paragraph being copied.

In the first case, the paragraph-name in the copied text must be identical to the paragraph-name in the data image containing the COPY statement. If the two names are identical, compilation continues with the first entry in the copied paragraph. If they are not identical, a diagnostic is issued and the library text is not incorporated in the compilation.

If the copied text begins with an Area B entry or if the Environment Division paragraph-name and its associated COPY statement are not contained on the same source statement, the redundancy checking described in the preceding paragraph is not performed. Rather, the entire library text is included in the compiled source. Note that this may result in duplicate Environment Division paragraph headers.

If the copied text begins with a comment, this image is included in the compiled source but the following image is used to check for the blank or nonblank Area A.

7.3. Library Entries For the Data Division

The entries in the library associated with the Data Division are of two types:

- Entries relating to the File, Report, or Communication Description portion of the Data Division. These are retrieved by the use of the COPY statement in an entry carrying an FD, SA, CD, RD, or SD level indicator. Record Descriptions may or may not be included with the FD, SA, CD, RD, or SD File description.
- Entries relating to the Record Description, Common-Storage, Working-Storage, or Linkage Sections of the Data Division. These are retrieved through the use of the COPY statement at the 01 level in the source program Record Descriptions.

Regardless of whether a Data Division COPY statement is invoked at a level indicator or an 01 level Record Description, a redundancy check is made between the source and the library text.

If the library text begins with a nonblank Area A and is not a comment, the level indicator or level number in the library text is compared to the level indicator or number in the data image containing the COPY statement. If the level indicators or numbers are identical, and the level indicator or number and the data-name in the library are terminated by a period, the source line will be assumed to be terminated by a period. If the level indicators or numbers are not identical, the COPY is terminated and a diagnostic is printed.

If the library text begins with an image which has a blank Area A or if the Data Division level indicator or number and its associated COPY statement are not contained on the same image, the redundancy check is not done and the entire library text is included in the compilation. Note that if it is the user's intent to initiate copying with data items

subordinate to an 01 level Record Description, it is essential that the first such item have its level number wholly in Area B. Also note that a copied Report Group Description which has text in Area A of its first line must contain both a level number and a data-name even though report group syntax allows the latter to be omitted.

If the library text begins with a comment, this image is included in the compiled source but the following image is used to check for the blank or nonblank Area A.

7.4. Library Entries for the Procedure Division

Each routine in the COBOL Library is composed of either one paragraph, identified by a paragraph-name, or one section, identified by a section-name. For purposes of copying this routine from the library, the paragraph-name or section-name is called a *procedure-name*.

Routines are retrieved from the library and copied into the source program through the use of the COPY statement in the Procedure Division. Then, at compile time the *procedure-name* that identifies the COPY statement replaces the library *procedure-name* that identifies the library routine. The format of a COPY clause written in the Procedure Division is as follows:

```
procedure-name copy-statement.
```

A COPY statement following a SECTION header must reference library text which begins with a section-name and a paragraph-name. The section-name in the library text will be skipped and the name of the section will be the source section-name which initiated the COPY.

If a COPY statement appears following a paragraph header and the copied text begins with a paragraph header, the paragraph-name in the library text will be skipped.

For either a section or paragraph COPY statement, library text following the skipped *procedure-name* (if any) will be incorporated in the compilation.

Section 8
Interprogram Communications

8.1. The ENTER Statement

Function

The ENTER statement transfers control from one object program to another within the same run unit.

Format 1

```
ENTER literal-1
```

Format 2

```
ENTER { FORTRAN   } literal-1
      { FD [ASM ] }

 [USING argument-1 [,argument-2]...[argument-31]]
```

Description

The *literal* must be a 1-to 12-character alphanumeric which is the name of the program-id or entry point of the program which is called.

When the G option is present in the Extra Options field of the @ACOB statement, parameters will be generated in the I-bank.

Numeric literal *arguments* are assumed to be fixed-point binary, right-justified in single or double precision as necessary. Floating-point *literals* are assumed to be double precision floating point. Nonnumeric *literals* are assumed to be DISPLAY (ASCII) unless the T option is set on the @ACOB processor call statement, in which case the nonnumeric *literal* will be DISPLAY-1 (Fieldata).

Arguments which are modified by a called program should not be specified as *literals* in the ENTER statement *argument* list. Not only would the modified value be unaddressable by the calling program upon return from the subroutine, but subsequent execution of calling statements which make use of *literals* might be incorrect.

Procedure-name *arguments* must not be used if a called program is written in COBOL. When numeric *literals* are used as *arguments*, procedure-names in the program should contain at least one alphabetic character to avoid a conflict between numeric *literal* and numeric procedure-name. If there is a conflict, COBOL will assume the argument is referring to the procedure-name.

If a calling program is itself called, it may pass, as an *argument*, data which was passed to it. That is, Linkage and Common-Storage Section identifiers may be included in the argument list of on ENTER statement.

Upon execution of an ENTER statement, control is transferred to the object code which corresponds to either the first nondeclarative statement in a program if a program-id-name is specified, or to the first statement of the named section or paragraph if that section-name or paragraph-name is listed as an entry point name for the called program.

Regardless of the name specified (program-id-name or entry-point-name), erroneous processing will occur if an ENTER in a nonrandom processing section or in a particular execution cycle of a random processing section causes either the calling or called program to be entered recursivelyù That is, only one ENTER for the same subprogram may be active until an EXIT PROGRAM corresponding to the first ENTER is executed.

8.2. COBOL Calling FORTRAN

A FORTRAN subprogram may be invoked by an ENTER statement in COBOL programs. Correct processing, however, is possible only if the arguments passed to the FORTRAN are processable by FORTRAN object code. In addition. variables and constants passed between COBOL and FORTRAN programs must be in internal formats acceptable to both processors. For example. FORTRAN complex variables have no direct counterpart in COBOL and COBOL ASCII (DISPLAY) items have no exact counterpart in FORTRAN.

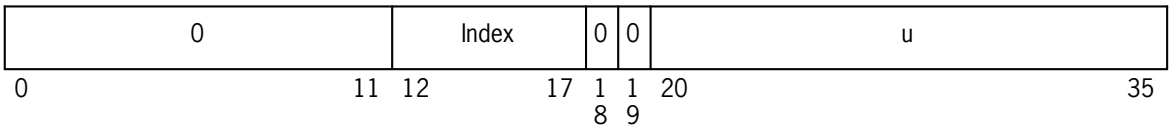
The subroutine calling sequences in COBOL and FORTRAN ore slightly different:

Calling Sequences	
COBOL	**FORTRAN**
IBJ$ X11, *subprogram-name* + C$B *subprog. number of arguments* +AC_1 +AC_2 . . . +AC_r	LMJ X11,*subprogram-name* + AF_1 + AF_2 . . . + AF_r *Walkback word*

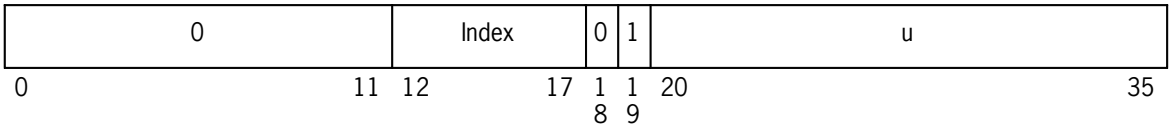
where:

1. IBJ$ is a collector XREF which maps into an LMJ or LIJ depending on the BDI of subprogram name.
2. C$B *subprog* is an entry point of the named subprogram's D-bank.
3. The number of arguments is an 18-bit binary number indicating the number of parameters which follow.

4. AC_1 is a COBOL argument address in the form:



5. AF_1 is a FORTRAN argument address in the form:



6. The walkback word is a word which is nonzero in S2 (FORTRAN uses this word to produce a diagnostic trace back to the main program through each subroutine used).

Thus, if a COBOL-generated calling sequence is used to invoke a FORTRAN subprogram:

1. The FORTRAN subprogram must have its first argument as a dummy parameter which cannot be used.
2. COBOL parameter addresses must not have an index for arguments AC_1 through AC_{r-1}. The COBOL compiler will generate acceptable argument addresses if the parameter is a 01-or 77-level data item in the Working-Storage or Common-Storage Section or if the parameter is a literal.
3. The last argument, AC_r, must be allocated but is not used unless an error occurs in the FORTRAN subprogram, in which case a guard mode will occur in attempting reference to the walkback word.

FORTRAN is not reentrant; therefore, be careful of asynchronous calling.

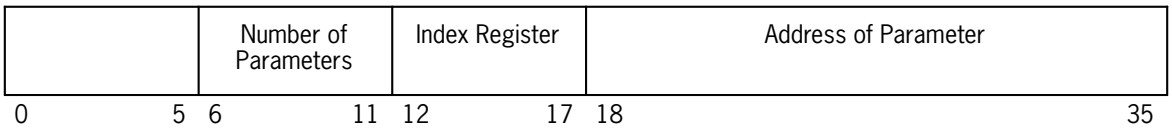
8.3. Downward Compatible Subprogramming

When format 2 of the ENTER statement is specified, parameters will be generated differently depending upon the phrase.

If the FD phrase is specified, the calling sequence will be as follows:

```
LMJ   X11,C$M701
LMJ   X11, subroutine
+     parameter-1
      .
      .
      .
+     parameter-n
LMJ   X11,C$M702
```

where *parameter-1* is as follows:



Note that ASCII COBOL will generate more than nine parameters but will issue a warning since Fieldata COBOL will not allow more than nine parameters. This in case a user wishes to pass this type of parameter sequence to a subroutine other than a Fieldata COBOL subroutine.

All other parameters will be the same as *parameter-1* omitting the number of parameters in S2.

For Fieldata COBOL, the ASCII COBOL RTL routine (C$M701) will be entered prior to transferring control to the subroutine and immediately after control is returned (C$M702). The purpose of such a routine will be to do PSR toggles to and from third-word mode and to save X6(KREG), X7(CREG), X9(SREG), R4-R7, R14-R15. This routine will also set up R-registers needed by Fieldata COBOL.

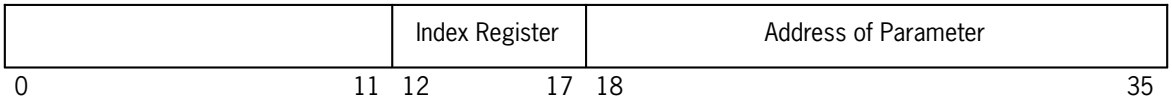
Since the Fieldata COBOL routines may not be reentrant, C$M701, C$M702 synchronize entry to the subroutines via a test and set (TS) instruction.

If the FORTRAN phrase is specified, the calling sequence will be:

```
LMJ  X11, subroutine
+ parameter-1
 .
 .
 .
+ parameter-n
NOP  S
```

where *S* is the line number of the COBOL source statement that caused generation of the calling sequence.

The parameter words are of the following form:



If the FD ASM phrase is specified, the calling sequence will be as follows:

```
LMJ   X11,C$M703
LMJ   X11, subroutine
      .
      .
      .
  ASM type parameters
      .
      .
      .
next instruction
```

The routine C$M703 will examine the parameters to the subroutine and create parameters that are compatible to those Fieldata COBOL does when calling an assembler subroutine using an ENTER SUBROUTINE REFERENCING statement. This routine will also toggle the PSR to and from third-word mode, synchronize calls to subroutines via a test and set (TS) instruction. and save X6(KREG), X7(CREG), X9(SREG), R4-R7, and R14-R15.

If the Common-Storage Section is used to share the main storage area between Fieldata COBOL and ASCII COBOL, it is necessary to insert a FILLER described as PIC X(6) USAGE IS DISPLAY-1 at the beginning of the ASCII COBOL Common-Storage Section to allow for the data allocation method used by Fieldata COBOL.

Section 9
Asynchronous Processing

9.1. The Saved-Area Description in the Data Division

Format

```
[,RECORD CONTAINS[ integer-1TO] integer-2CHARACTERS]
```

Description

The RECORD CONTAINS clause is ignored and serves as documentation only.

9.2. Procedure Division

9.2.1. The PROCESS Statement

Function

The PROCESS statement initiates a set of USE procedures under the control of an Asynchronous Control System (ACS).

Format

```
PROCESS section-name[[FROM  identifier]USING record-name]
```

Syntax Rules

1. The parameter *section-name* specified in the PROCESS statement must be the name of a USE procedure defined in the Declarative Section with a USE FOR RANDOM PROCESSING clause in its header section. It identifies the Random Processing Section to be executed.
2. The USING phrase is required for programs containing more than one Saved-Area Description.
3. The parameter *record-name* must be the name of a level 01 Record Description entry which is subordinate to a Saved-Area description entry (SA) defined in the File Section.

General Rules

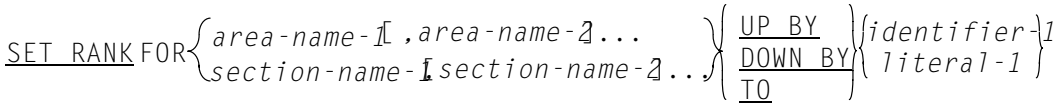
1. The PROCESS statement causes the named Random Processing Section to be executed as an asynchronous USE cycle. Actual commencement of execution for the initiated cycle is a function of the Asynchronous Control System. Control for the initiating cycle is returned to the next statement following the PROCESS statement.
2. The cycle which contains the PROCESS statement will be suspended until another execution of the named Random Processing Section is completed if activation of this cycle would exceed the maximum number of cycles stated in the FOR *integer-2* CYCLES clause of the named Random Processing Section. See the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version).
3. When the optional phrase FROM is used, moving an *identifier* to the copy of the Saved-Area allocated for the cycle initiated by the PROCESS statement takes place in accordance with the rules specified for the MOVE statement without the CORRESPONDING option.
4. The PROCESS statement causes the allocation of one copy of the named Saved-Area and one copy of each PROCESSING MODE IS RANDOM or PROCESSING MODE IS RANDOM CD stated in the USING clause of the USE FOR RANDOM PROCESSING clause for the named Random Processing section to be dedicated to the exclusive use of the cycle initiated at execution of the PROCESS statement. Saved-Area copies are released for reassignment upon completion of the cycle to which they were dedicated. If no copies of the SA or if a record area of a PROCESSING MODE IS RANDOM file are available, the cycle which has the PROCESS statement is suspended until a copy of the record-area or SA is made available by completion of another cycle.
5. Exclusive use of shared data items can be assured only through explicit embedding of appropriate LOCK and UNLOCK statements throughout the source language programs. Data items which occupy portions of a word in conjunction with a shared data item must also be treated as an extension of the shared data item. It is therefore advisable that shared data items be either implicitly or explicitly word synchronized.

9.2.2. The SET Statement

Function

The SET RANK statement dynamically changes the value of the RANK associated with a Random-Processing Section or Saved-Area description.

Format



Description

All *section-name* clauses must be names of an out-of-line procedure in the declarative section which contains a USE FOR RANDOM PROCESSING statement.

Area-name clauses must be names of Saved-Area description entry (SA) defined in the File Section of the Data Division.

All *identifiers* and *literals* must be positive integers.

The RANK of each RP-section and Saved-Area is changed according to the SET RANK statement. Each cycle priority for all active cycles is then recalculated as the sum of the RANK of the RP-Section and RANK of the Saved-Area for the cycle. The Asynchronous Control System locks out any other access to ACS until the recalculation of all cycles is done.

Section 10
Processor Call Statement

For a complete description of the @ACOB Processor Call statement, see the OS 1100 ASCII COBOL Programming Reference Manual, UP-8582 (current version). The following options are not described in that manual:

Table 10-1. @ACOB Processor Call Options

Option	Description
M	Ignore MONITOR statements.
T	Reverse DISPLAY and DISPLAY-1; implicit and DISPLAY usage mean Fieldata. Reverse COMPUTATIONAL and COMPUTATIONAL-4. This option does not affect data items described with a PICTURE clause containing 1's (exact binary). This option does not affect the special registers.

Table 10-2. @ACOB Processor Call Extra Options

Extra Option	Description
E	Process American National Standard 1968 COPY statements and library text only. American National Standard 1974 COPY statements and library text cannot be processed with this option.
G	Generate the parameter list for the CALL statement in I-bank. When the parameters are generated in the I-bank, the called program cannot be banked.
O	Process American National Standard 1968 CODE clause, which implies fixed-length images. American National Standard 1974 CODE clause, which implies variable-length images, will not be processed.